UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

End of degree project

Bachelor's degree in Mechanical Engineering

State feedback control and Luenberger observer design for a differential-drive mobile robot

Report and Annexes

Author: Marc Murcia Matute
Supervisor: Arnau Dòria Cerezo
Co-Supervisor: Victor Repecho

Call: June 2019

Resum

Aquest projecte proposa un nou controlador per a un robot que segueix una línia, basat en un model matemàtic que treballa només amb variables locals del robot, la distancia a la línia i l'angle de desviació de l'eix del robot respecta la tangent de la corva. El controlador dissenyat i implementat des d'aquest model és un controlador de representació en espai d'estats amb la utilització d'un observador de Luenberger, per estimar l'angle de desviació. Aquest nou controlador aporta una millora del rendiment del vehicle en qualsevol circuit movent-se cap endavant. A més, també funciona amb unes limitacions cap enrere, sense haver d'instal·lar un nou sensor de línia a la part posterior, aportant un punt inicial per a la completa implementació d'un controlador que funciona en ambos sentits.

Durant el transcurs d'aquest treball s'han dut a terme diferents simulacions numèriques, i són mostrades en el projecte per a la defensa del nou controlador. També s'ha implementat en un robot, en el qual s'ha desenvolupat el codi en C i està adjunt en l'annex.

Resumen

En este proyecto se propone un nuevo controlador para un robot que sigue una linea, basado en un modelo matematico que trabaja solo con variables locales del robot, la distancia a la linea y el ángulo de desviación del eje del vehículo respecto a la tangente de la curva. El controlador diseñado e implementado desde este modelo es un controlador por representación de espacios de estados con la utilización de un observador de Luenberger, para la estimación de ángulo de desviación. Este controlador aporta una mejora en el rendimiento del vehículo en cualquier circuito dado al moverse hacia adelante. Además, funciona con unas limitaciones en marcha atrás, sin la necesidad de instalar un nuevo sensor de linea en la parte posterior, siendo un punto de inicio para la completa implementación de un controlador que funciona en ambos sentidos.

Durante el transcurso de este trabajo se han ejecutado diferentes simulaciones numéricas y son mostradas en el proyecto para la defensa del controlador. También se ha implementado en un robot, en el cual el código ha sido escrito en C y esta adjunto en el anexo.

Abstract

This end of degree project suggest a new control algorithm for a differential-drive mobile robot, based on a mathematical model that works only use robot local variables, the distance from the line and the deviation angle between the robot axis and the tangent of the path. A state-space feedback controller with a Luenberger observer, for the estimation of the deviation angle, had been designed from the model. This new control algorithm enhance the robot response in any given track. Moreover, allows the vehicle for backwards move under certain constrains, without the need of installing a new sensor in the back part. Being a initial point for the complete development of a control algorithm able to move in both directions.

During the development of this project the controller was tested in several numerical simulations, this simulations are exposed and explained supporting the new solution. As well the code implemented in the robot, written in C, is annexed to the memory.

Acknowledgements

First I would like to acknowledge my tutors Arnau Dòria-Cerezo and Vítor Repecho del Corral, without who I would not had been able to develop this research project. Arnau giving me always support in the theoretical side and Victor helping in any problem I could have in the implementation process in the laboratory.

Secondly, I want to thanks all the students that developed the previous projects, establishing the basis to evolve my project.

Finally all my close people that supported me and encourage during its development, making the whole process easier.

Glossary

- **ARM** Microprocessor family architecture of the RISC (Reduced Instruction Set Computer) type. Most used in embedded systems due to its low cost and low energetic consume.
- **PWM** is the pulse-width modulation of the signal in order to transmit data or control the energy provided.
- An **encoder** is a sensing device that transform data from a format to another.
- An **Integrated Development Enviroment (IDE)** is a computer application used to facilitate the programming of software.
- An **Analog to Digital Converter (ADC) or Digital to Analog Converter DAC** is a tool that converts the electric signal.
- **State-space representation** is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations.
- The **state variables** are the smallest possible subset of system variables that can represent the entire state of the system at any given time.
- **State observer** is a system that provides an estimate of the internal state of a given real system, from measurements of the input and output of the real system.
- A **Central Processing Unit (CPU)** is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output operations specified by the instructions.
- A **Microcontroller** is a small computer on a single integrated circuit. Containing one or more CPUs along with memory and pogrammable input/output peripherials.
- An **embedded system** is a controller with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.
- **C** is a general-purpose, imperative computer programming language supporting structured programming, lexical variable scope, and recursion, while a static type system prevents unintended operations.

Contents

1 | Introduction

This project is the continuation of several end of bachelor projects all in the IOC (Institute of Industrial and Control Engineering) line of investigation about two wheeled vehicle robots, aiming to experimentally simulate different scenarios as platooning, autonomous vehicles, overtaking manoeuvres, automatic vehicle lane changes and shock-wave traffic jams.

In the previous projects, the basic steps to the final objectives were notably achieved. While Iván Prats assembled, modelled and programmed the first concept of the robot in his thesis [1], Antoni Riera developed a Wi-Fi communication system in order to command and exchange information with the vehicles using a computer [2], Albert Costa united the previous mentioned papers and did some simulations, using the results to improve the robot's controllers [3]. Finally, Oriol Torta assembled and documented a new robot and designed and implemented a current control for the motors [4].

This particular project starts on the basis of an article called 'Sliding mode control of a differential-drive mobile robot following a path' [5], in order to implement a more precise trajectory control algorithm for the vehicle developed in the previous mention projects.

1.1 Objectives

In order to achieve the main target, an enhancement of the trajectory control algorithm, the project was segmented into some minor objectives that must be realised.

- Analysis and study of the suggested mathematical model [5].
- Design of a new trajectory controller based on the mathematical model, a state-space feedback control. Learn about state-space controllers [6].
- Design of a Luenberger observer. Learn about observers [6].
- Recreation of simulations regarding the response of the proposed control algorithm in different scenarios.
- Implementation to the vehicle using a STM32F4-Discovery Board as microcontroller. Learn C programming language.

2 | Hardware description

Even thought in this project there is no focus on the design of the robot and its construction, in this chapter are mentioned the different components composing the vehicle and a brief description and information of the most important element is given.

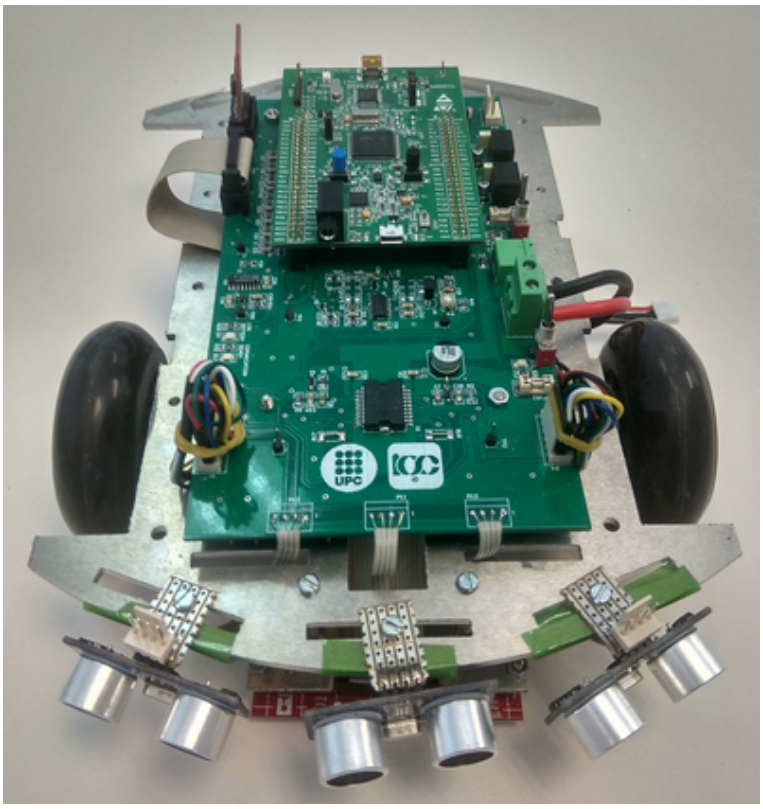

Figure 2.1: Differential-drive mobile robot

2.1 STM32F4 Discovery Board

The control of the motors, acquisition and manipulation of all the collected signals by the vehicle sensors require a powerful microcontroller. The STM32F4-Discovery is an economic microprocessor that includes a STM32F407G microcontroller based on ARM-Cortex architecture, along with some useful peripherials like timers, PWM channels and ADC/DAC converters, together with all the needed components for a proper functionality, developed by STMicroelectronics. Widely known as cheap and accessible device, due to the availability of many freeware tools to configure and program without high programming skills.

STM32F407G-DISC1 features	
CPU	32-bit ARM Cortex-M4 with Floating Point Unit(FPU) core
Flash memory	1 Mbyte
RAM memory	192 Kbyte in an LQFP100 package
Clock rate	168 MHz
Timmers	Twelve general-purpose I/O 16-bit timers (GPIO) including two PWM timers for motor control and two general-purpose 32-bit timers.
Communication interfaces	2 Universal Asynchronous Receiver/Transmitter (UART)
Debug tool	ST-LINK/V2-A embedded debug tool with re-enumeration capability and three different interfaces: • Debug port • Virtual Com port • Mass storage
Board power supply	Through USB bus or from an external 5V supply voltage
External application power supply	3 V and 5 V
Accelerometer	LIS3DSH ST MEMS 3-axis accelerometer
Audio sensor	MP45DT02 ST-MEMS audio sensor omni-directional digital microphone
Audio	CS43L22 audio DAC with integrated class D speaker driver
LEDs	• LD1 (red/green) for USB communication • LD2 (red) for 3.3 V power on • Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue) • 2 USB OTG LEDs LD7 (green) VBUS and LD8 (red) over-current
Buttons	One user and one reset push-button
Other peripherials	• Three 12-bit ADCs (Analog to Digital Converter) • Two DACs (Digital to Analog Converter) • A low-power RTC (Real Time Clock)

Table 2.1: STM32F4 features [7]

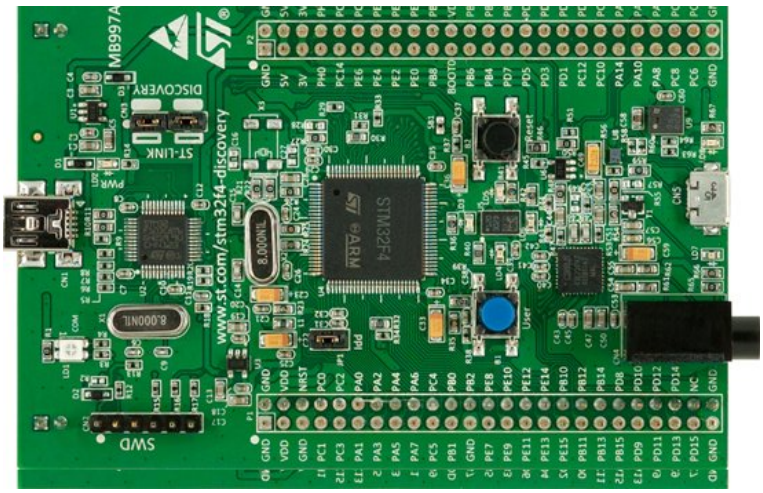

Figure 2.2: STM32F4-Discovery Board [4].

2.1.1 Programming tools

As mentioned before, the STM32F4 Discovery Board has many possible developing tools for programmers. To develop the C code of the project has been employed a free IDE (Integrated Development Environment) for STM32 microprocessors, System Workbench for STM32, developed by Ac6 Tools in partnership to STMicroelectronics, which consist on a set of Eclipse plug-ins. The installation process and all the information necessary to correctly evolve the project was extracted from an online community, OpenSTM32 Community [8].

2.1.2 Used peripherials

From the wide range of useful peripherials that the microcontroller STM32 offers to its user, a few were selected to develop this project:

- One Advanced-control Timer (TIM1), used to generate a Pulse-Width Modulation (PWM) signal of the motor driver, controlling the DC motors. It also governs the interruption routine, containing the vehicle's control. This timer has a 168MHz counter frequency and the interruption routine is triggered with a $50\mu s$ period.

- One General-purpose Timer (TIM2), in Input Capture mode used as a capture timer for the pulses of the 3 proximity sensors.

- Two General-purpose Timers (TIM3 and TIM4) to the the wheels encoder's frequency. Configured in Input Capture direct mode, for reading the pulse signals of the photo-interrupters and assigning them the corresponding counter clock value, for a posterior calculation of the wheels speed. TIM3 is used to read the left encoder(left wheel) and TIM4 to read the right encoder(right wheel).

- One General-purpose Timer (TIM7), used for synchronisation for the wheel speed measurement

- Two General-purpose Timers (TIM10 and TIM12) in charge of managing the ultrasonic sensor. TIM10 generates the trigger signal for the proximity sensors, while TIM12 set in Input Capture mode is used as a capture timer for the proximity sensors.
- One 12 bit Analog to Digital Converter (ADC1) with 4 input channels. Two used for reading the signal of the line sensor. And the other pair for acquiring the values of the current sensor. All data acquisition is done through Direct Memory Access (DMA) allowing the acquisition of the ADC's data simultaneously with other operations the CPU must do. Once the acquisition is done the CPU receives a interrupt signal from the DMA controller, using that method we can reduce the data acquisition time.
- One Digital to Analog Converter (DAC), used to send digital information to the PC in order to graph and analyse its behaviour.

2.2 Other components

- auxiliary board with resistances, capacitors, diodes, switchers and other common electronic, assembled by the IOC-UPC
- 1 two-wheeled chassis
- 2 DC bidirectional motors with its wheels and rotatory encoders
- 1 multi-directional roller ball
- 3 ultrasonic sensor HC-SR04
- 1 motor driver L298N
- 1 line and light sensor LRE-F22
- 2 photo-interrupters RPI-574
- 1 WiFi module ESP8266
- 1 regulator LM317
- 1 operational amplifier
- 1 SN74HC04n integrated circuit with 4 inverters
- 1 battery

3 | Modelling and design of the control

During this chapter the mathematical model suggested in [5] is develop and analysed for the design of the state-feedback controller governed by a Luenberger observer. This mathematical model simplifies the system to two variables, the angle between the line tangent and the vehicles axis and the distance from the track, being taken as state-space variables of the controller. Moreover, the observability and controllabilty of the system is deeply studied.

3.1 Model of a two-wheel differential-drive mobile robot tracking a path

The aim of the problem consist in a differential-drive robot able to follow a path with a predetermined longitudinal speed v_x. In this section the trajectory pursued by the vehicle will be parametrized to obtain the kinematic model, with which we will design the controller.

When the two-wheeled vehicle perfectly follows the path, point P, where an optical line sensor is installed, is on the curve $\sigma(q)$, that represents the desired path of the robot.

For the vehicle to perfectly follow the track Figure 3.1 one condition is required, that the distance between P_q and P (where the line sensor is installed) in the y_m direction, local y axis of the vehicle, must be aimed to be $d = \overline{PP_q} = 0$.

The kinematic model of the unicycle-type mobile robot with respect to point P_m (located in the middle of the wheels axis, chosen because its the central point of the vehicles movement) is given by the following equations:

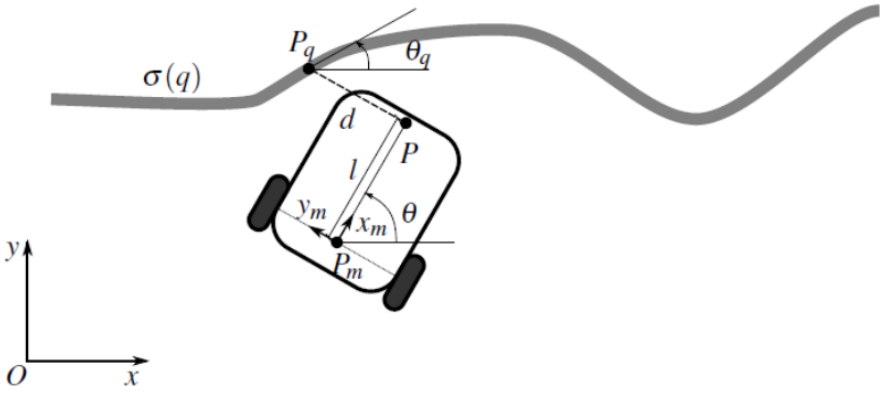

Figure 3.1: Two-wheel differential drive mobile model. Source: [5]

$$\dot{x}_m = v\cos(\theta) \tag{3.1}$$

$$\dot{y}_m = v\sin(\theta) \tag{3.2}$$

$$\dot{\theta} = -u \tag{3.3}$$

with v being the vehicle movement respect the x local reference coordinate of the vehicle and u the robots angular velocity around P_m:

$$v = \frac{r}{2}(\omega_l + \omega_r); \quad u = \frac{r}{2R}(\omega_l - \omega_r) \tag{3.4}$$

where x_m, y_m are the local reference coordinates, θ_q is the angle between the tangent of the curve $\sigma(q)$ in the point P_q , and the global x coordinate axis (orientation of the vehicle), r is the wheels radius, ω_l, ω_r are the angular velocity of the left and right wheel respectively and R is the distance between both wheels.

From figure 3.1 we can observe that

$$\overline{OP_q} = \overline{OP_m} + \overline{P_m P_q} \tag{3.5}$$

using the coordinates of its vectors respect to the global coordinate system, for $\overline{OP_q}$ being $(\sigma_x(q), \sigma_y(q))$, $\overline{OP_m} = (x_m, y_m)$ and for $\overline{P_m P_q}$ being the distance between the middle wheel's axis and the optical sensor, l, and the distance d, multiplied by the rotation matrix, we can write it as

$$\begin{pmatrix} \sigma_x(q) \\ \sigma_y(q) \end{pmatrix} = \begin{pmatrix} x_m \\ y_m \end{pmatrix} + R(\theta) \begin{pmatrix} l \\ d \end{pmatrix} \tag{3.6}$$

where $R(\theta)$ is $\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$.
By differentiating (3.6) and using (3.1), (3.2) and (3.3) we determine the motion of the vehicle in terms of d, q (velocity on the path) and θ (angle between the x_m axis and the global x axis):

$$\dot{d} = lu - (v + du)\tan(\theta - \theta_q) \tag{3.7}$$

$$\dot{q} = \frac{1}{\cos(\theta - \theta_q)}(v + du)l \tag{3.8}$$

$$\dot{\theta} = -u \tag{3.9}$$

where $\frac{\partial \sigma_x}{\partial q} = \cos(\theta_q)$ and $\frac{\partial \sigma_y}{\partial q} = \sin(\theta_q)$ were used. As mentioned before, the angle θ between the x global axis and the x_m local axis of the vehicle, and the angle θ_q between the tangent of the path at P_q and the global axis x are of no interest, instead when the robot is in equilibrium the difference of this angles must be zero, so that the vehicle is always pointing in the same direction than the track. Given that, the deviation angle is defined as $\theta_e = \theta - \theta_q$ modifying the dynamics

to

$$\dot{d} = lu - (v + du)\tan(\theta_e) \tag{3.10}$$

$$\dot{q} = \frac{1}{\cos(\theta_e)}(v + du) \tag{3.11}$$

$$\dot{\theta}_e = -u - \frac{c(q)}{\cos(\theta_e)}(v + du) \tag{3.12}$$

where $c(q) = \frac{\partial \theta_q}{\partial q}$ is the curvature of the path $\sigma(q)$. Consider that $\frac{\partial \theta_q}{\partial q}\frac{\partial q}{\partial t} = \frac{\partial \theta_q}{\partial t} = \dot{\theta}_q$.
Assuming that v is designed such that $\dot{q} = v_x$ is guaranteed, a predetermined longitudinal speed. Therefore, the remaining dynamics are

$$\dot{d} = lu - v_x \sin(\theta_e) \tag{3.13}$$

$$\dot{\theta}_e = -u - c(q)v_x \tag{3.14}$$

where the curvature, $c(q)$, is treated as a disturbance.
The responses at the equilibrium point are studied, in order to determine the optimal control values and possible restrictions of the mathematical model. From (3.13) and (3.14) we get that the required control value, u^*, and the corresponding deviation angle, θ_e^* , that ensures $\dot{d} = \dot{\theta}_e = 0$ are

$$u^* = -cv_x \tag{3.15}$$

$$\theta_e^* = \arcsin(-cl) \tag{3.16}$$

which gives as a maximum curvature constraint of

$$c < c_{MAX} = \frac{1}{l}. \tag{3.17}$$

The mathematical model used is a non-linear system, a system in which change of the output is not proportional to the change of the input, that is caused by the influence of the sinusoidal function. Nevertheless, the system can be linearised using the Taylor series expansion, doing that change we are assuming an error between the real system, the non-linear model, and the linearised model used for the controller. The linear approximation system, around the working points, $d^* = 0$, becomes

$$\begin{pmatrix} \dot{d} \\ \dot{\theta}_e \end{pmatrix} = \begin{pmatrix} 0 & -v_x \cos(\theta_e^*) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} d \\ \widetilde{\theta}_e \end{pmatrix} + \begin{pmatrix} l \\ -1 \end{pmatrix} \widetilde{u} \tag{3.18}$$

$$y = (1\ 0) \begin{pmatrix} d \\ \widetilde{\theta}_e \end{pmatrix} \tag{3.19}$$

where $\theta_e^* = \arcsin(-cl)$, $\widetilde{\theta}_e = \theta_e - \theta_e^*$ and $\widetilde{u} = u - u^*$. Remember that curvature was decided to be treated as a disturbance.

The dynamics of the system can be written as a transfer function of the input/output

$$Y(s) = G(s)U(s) \tag{3.20}$$

$$G(s) = C(sI - A)^{-1}B + D \tag{3.21}$$

with $A = \begin{pmatrix} 0 & -v_x \cos(\theta_e^*) \\ 0 & 0 \end{pmatrix}$; $B = \begin{pmatrix} l \\ -1 \end{pmatrix}$; $C = (1\ 0)$
we obtain the following transfer function of the linearised system

$$Y(s) = \frac{ls + v_x \cos(\theta_e^*)}{s^2} U(s). \tag{3.22}$$

3.2 Controller design

In this section a state-feedback controller is going to be design for a differential-drive mobile robot. We added the integral action to correct the control based on the accumulated error over the time.

$$u = -k_i z - k_d d - k_\theta \widetilde{\theta_e} \tag{3.23}$$

where $\dot{z} = y = d$.
Linearising our system, (11), with the added integral to

$$\begin{pmatrix} \dot{d} \\ \dot{\theta}_e \\ \dot{z} \end{pmatrix} = \begin{pmatrix} 0 & -v_x \cos(\theta_e^*) & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} d \\ \theta_e \\ z \end{pmatrix} + \begin{pmatrix} l \\ -1 \\ 0 \end{pmatrix} u \tag{3.24}$$

Combining (3.23) and (3.24), introducing the control action to the state space plant, our system transforms to

$$\begin{pmatrix} \dot{d} \\ \dot{\theta}_e \\ \dot{z} \end{pmatrix} = \left(\begin{pmatrix} 0 & -v_x \cos(\theta_e^*) & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} - \begin{pmatrix} l \\ -1 \\ 0 \end{pmatrix} (k_d\ k_\theta\ k_i) \right) \begin{pmatrix} d \\ \theta_e \\ z \end{pmatrix} \tag{3.25}$$

Yielding in the following state matrix

$$A = \begin{pmatrix} -lk_d & -lk_\theta - v_x \cos(\theta_e^*) & -lk_i \\ k_d & k_\theta & k_i \\ 1 & 0 & 0 \end{pmatrix} \tag{3.26}$$

whose characteristic polynomial is

$$\lambda^3 + (k_d l - k_\theta)\lambda^2 + (k_i l + k_d v_x \cos(\theta_e^*))\lambda + k_i v_x \cos(\theta_e^*) = 0 \tag{3.27}$$

Using the Routh-Hurwitz criterion for third order systems, can be determined the conditions that must be granted for the controller to be stable:

$$k_d l - k_\theta > 0 \tag{3.28}$$

$$k_i v_x \cos(\theta_e^*) > 0 \tag{3.29}$$

$$(k_i l + k_d v_x \cos(\theta_e^*))(k_d l - k_\theta) > k_i v_x \cos(\theta_e^*) \tag{3.30}$$

3.3 Observer design

The controller proposed requires the knowledge of θ_e. Consequently, a Luenberger observer must be design in order to estimate its value. Analysing the observability condition of the system (3.18), (3.19), we obtain the following observability matrix

$$W_0 = \begin{pmatrix} 1 & 0 \\ 0 & -v_x \cos(\theta_e) \end{pmatrix} \tag{3.31}$$

which is full rang as $v_x \neq 0$. Therefore, the system is observable whenever the vehicle is moving. The proposed Luenberger observer is

$$\begin{pmatrix} \dot{\widetilde{d}} \\ \dot{\widetilde{\theta}}_e \end{pmatrix} = \begin{pmatrix} 0 & -v_x \cos(\theta_e^*) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \widetilde{d} \\ \widetilde{\theta}_e \end{pmatrix} + \begin{pmatrix} l \\ -1 \end{pmatrix} \widetilde{u} + \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} (d - \widetilde{d}) \tag{3.32}$$

yielding to a state matrix of the observer

$$A_o = \begin{pmatrix} -L_1 & -v_x \cos(\theta_e^*) \\ -L_2 & 0 \end{pmatrix} \tag{3.33}$$

described by a characteristic polynomial

$$\lambda^2 + L_1 \lambda - L_2 v_x \cos(\theta_e^*) = 0 \tag{3.34}$$

Using the Routh-Hurwitz criterion for second order polynomials, the observer will be stable if $L_1 > 0$ and $-L_2 v_x \cos(\theta_e^*) > 0$.
This observer was design separately from the controller, following the separation principle that suggest that if a stable observer and a stable controller for a linear time-invariant system are designed, the combination of those is also stable. Was possible to apply this principle because the system was linearised.

3.3.1 Error of the proposed controller

Taking the curvature as a disturbances, neglecting it, generates an error in the estimated states by the Luenberger observer. In the following system the neglected factor was introduced to check the

amount of error produced, when the $c = 0$ and $c \neq 0$

$$\begin{pmatrix} \dot{\widetilde{d}} \\ \dot{\widetilde{\theta}}_e \end{pmatrix} = \begin{pmatrix} 0 & -v_x \cos(\theta_e^*) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \widetilde{d} \\ \widetilde{\theta}_e \end{pmatrix} + \begin{pmatrix} l \\ -1 \end{pmatrix} \widetilde{u} + \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} (d - \widetilde{d}) + \begin{pmatrix} 0 \\ -cv_x \end{pmatrix} \tag{3.35}$$

Checking the equilibrium point $\dot{\widetilde{d}} = \dot{\widetilde{\theta}}_e = 0$

$$\widetilde{u^*} = L_2(d - \widetilde{d}) - cv_x \tag{3.36}$$

$$\widetilde{\theta_e^*} = \frac{l\widetilde{u} + L_1(d - \widetilde{d})}{v_x \cos(\theta_e^*)} \tag{3.37}$$

combining both required values that ensure the equilibrium, the error produced when $c = 0$ can be defined as $e = x_{c\neq 0} - x_{c=0}$

$$e_{\widetilde{u^*}} = -cv_x = e_{u*} \tag{3.38}$$

$$e_{\widetilde{\theta_e^*}} = \frac{-cl}{\cos(\theta_e^*)} = \tan(\theta_e^*) \tag{3.39}$$

This produces a considerable error in the estimation of the angle that if curvature was known could significantly optimise the controller. Modifying the control action and observer to

$$u = -k_i z - k_d d - k_\theta \widetilde{\theta}_e - cv_x \tag{3.40}$$

$$\begin{pmatrix} \dot{\widetilde{d}} \\ \dot{\widetilde{\theta}}_e \end{pmatrix} = \begin{pmatrix} 0 & -v_x \cos(\theta_e^*) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \widetilde{d} \\ \widetilde{\theta}_e + \tan(\theta_e^*) \end{pmatrix} + \begin{pmatrix} l \\ -1 \end{pmatrix} \widetilde{u} + \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} (d - \widetilde{d}) \tag{3.41}$$

4 | Simulations

The control algorithm and the observer design in the previous chapter have been tested in numerical simulations using Matlab and Simulink. The code and blocks are shown in the Annex.
Theoretically, the suggested control algorithm could work on both directions, forwards and backwards, because of the knowledge of the deviation angle. Given that, in the numerical simulations were added some cases to test the control and observer algorithms when the vehicle moves backwards
Graphs data:
(A) x, y plane pot: distance sensor(blue dot), right wheel (green dot) and left wheel (red dot).
(B) distance to the path. d, respect to x.
(C) distance to the path, d, respect to time.
(D) control action, u.
(E) wheel's speeds, ω_r, ω_l.
(F) estimated deviation angle $\widetilde{\theta}_e$ (in red), with respect to the real, θ_e (in blue), in radians.

4.1 Initial description

The controller was tested in four different tracks, with a constant curvature $c = 0$ Track 1, Track 2 $c \neq 0$ with a 30° angle,Track 3 with a constant curvature $c \neq 0$ (forward motion has been tested in three different curvatures, with radius $0.75m$, $0.45m$ and $0.15m$, and backwards with radius $0.75m$ and $0.45m$) and Track 4 with a not constant curvature $c \neq ct$. The robot parameters for all tests are $l = 0.07m$, $R = 0.05m$ and $r = 0.02m$, with a desired longitudinal speed of $v_x = 0.3m/s$ or $v_x = -0.3m/s$. The control characteristics moving forwards are a settling time of $t_s = 3s$, and the observer parameters of $L_1 = 50$ and $L_2 = -360$. And the inital conditions are $\theta_0 = 0.2rad$ and $d = 0.03m$. Moving backwards a settling time of $t_s = 5.5s$, and with the observer gains $L_1 = 50$ and $L_2 = 1650$. The initial conditions are $\theta_0 = 0.15rad$ and $d = 0.015m$. Moreover, in the Track 3 is compared the response when the curvature unknown, equations (3.23) and (3.18), versus when the curvature is known (3.40) and (3.41), in that case the observer gain L_2 changes from $L_2 = -710$ (forwards) to $L_2 = 730$ (backwards).

4.2 Analysis

From the simulation results, both control and observer algorithm work well when moving forward, stabilising in the desired time and estimating the real deviation angle, Figures 4.3, 4.4 and 4.5, with notable success in both cases, when the curvature is known and unknown, supporting the treatment of the curvature as a disturbance and neglecting its influence to the mathematical system. Whenever is possible, the knowledge of the curvature becomes more important when it is approaching c_{MAX} (3.17) giving a faster and more precise response, see Figure 4.5. When the curvature is not constant, $c \neq ct$, the control algorithm and observer still is functional, stabilising

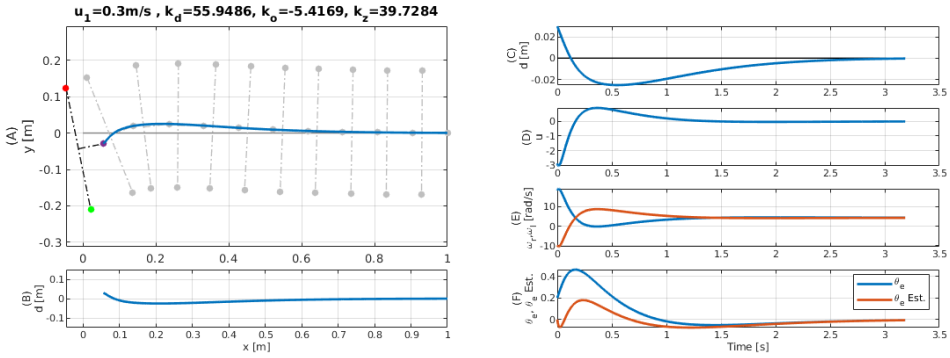

Figure 4.1: Track 1 moving forward

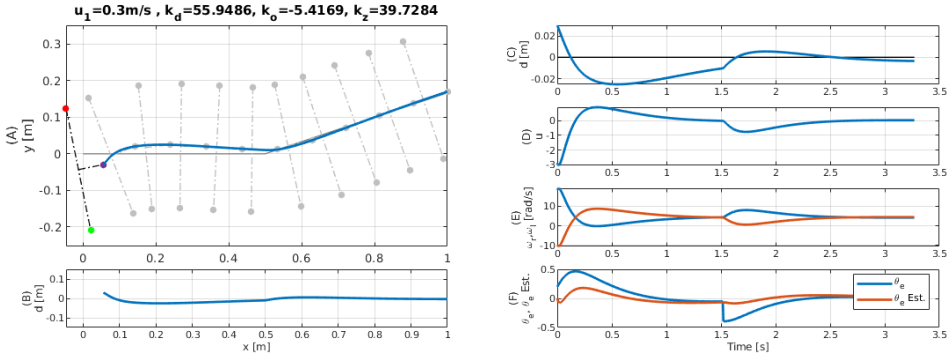

Figure 4.2: Track 2 moving forward

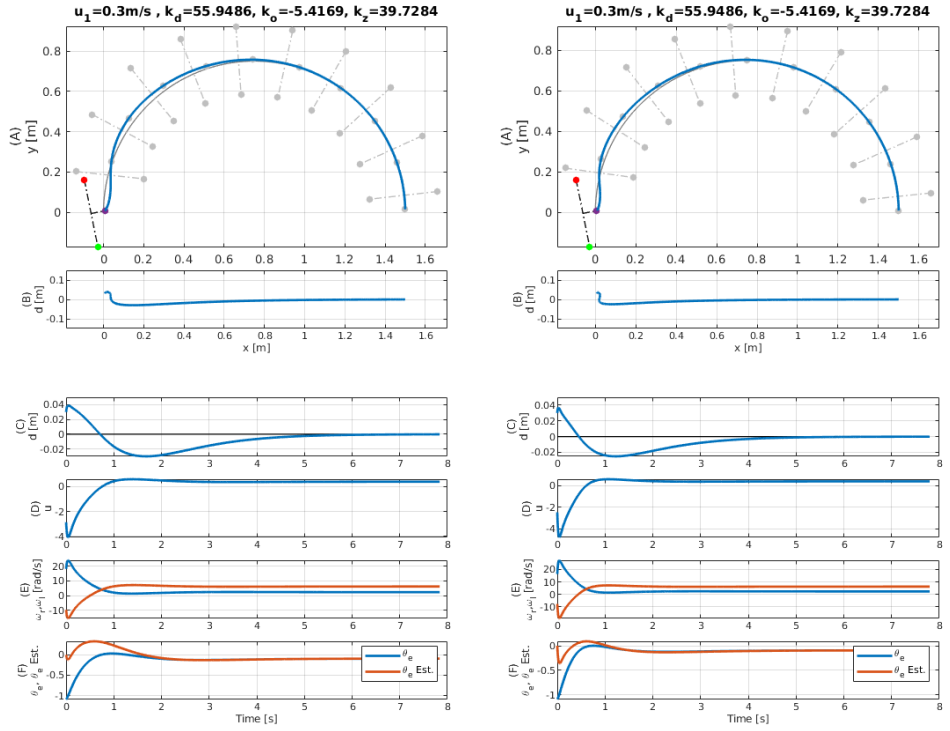

Figure 4.3: Track 3 with radius $r = 0.75m$, left unknown curvature and right $c = 1/0.75m^{-1}$

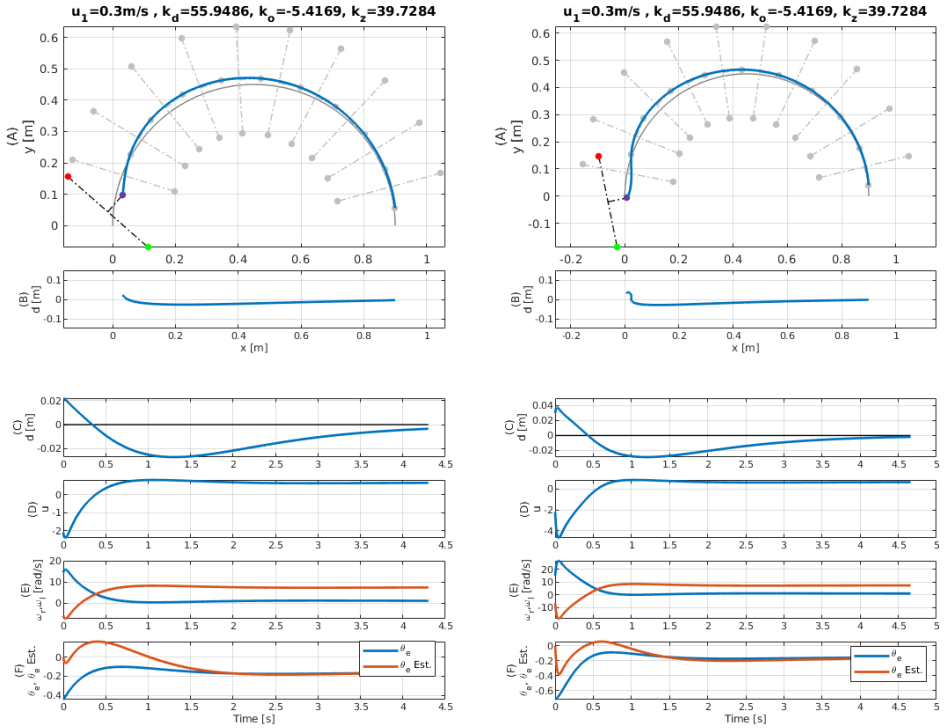

Figure 4.4: Track 3 with radius $r = 0.45m$, left unknown curvature and right $c = 1/0.45m^{-1}$

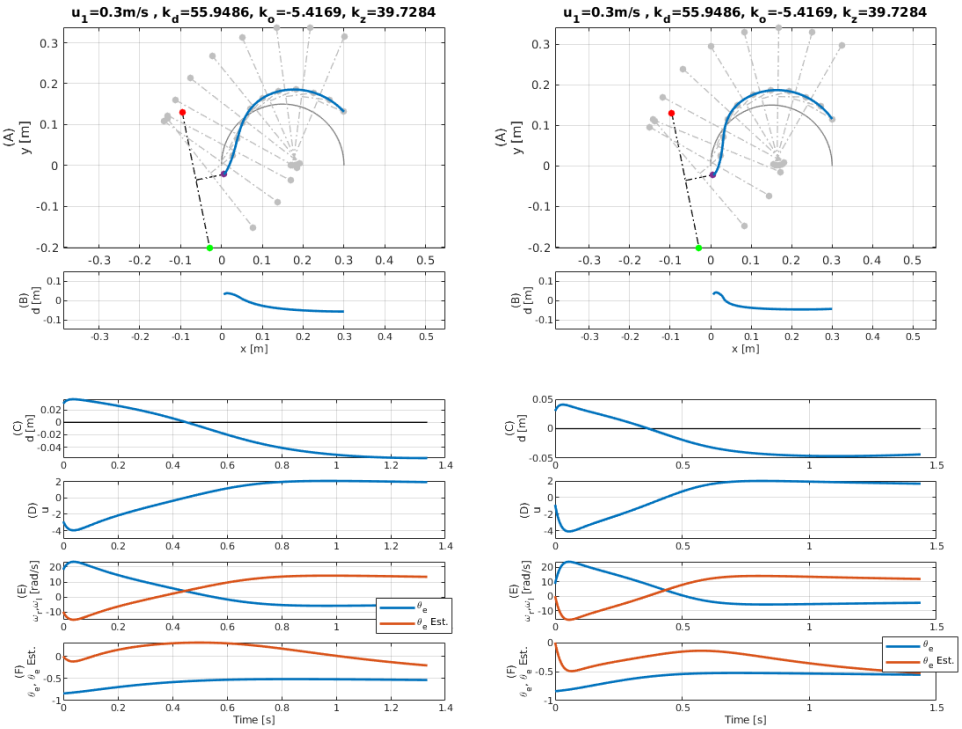

Figure 4.5: Track 3 with radius $r = 0.15m$, left unknown curvature and right $c = 1/0.15m^{-1}$

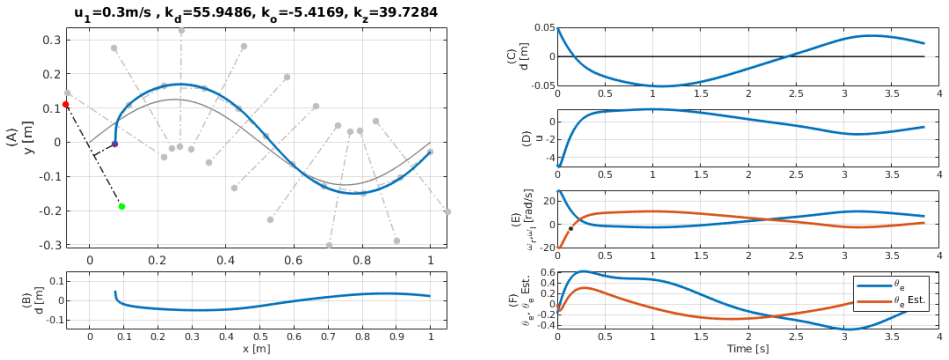

Figure 4.6: Track 4 moving forward

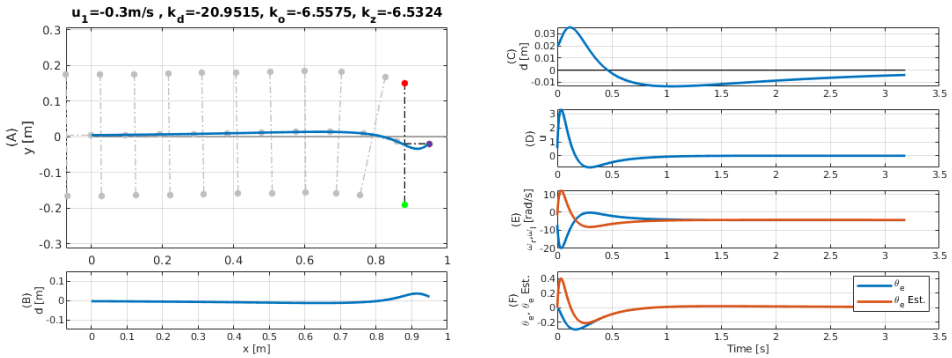

Figure 4.7: Track 1 moving backwards

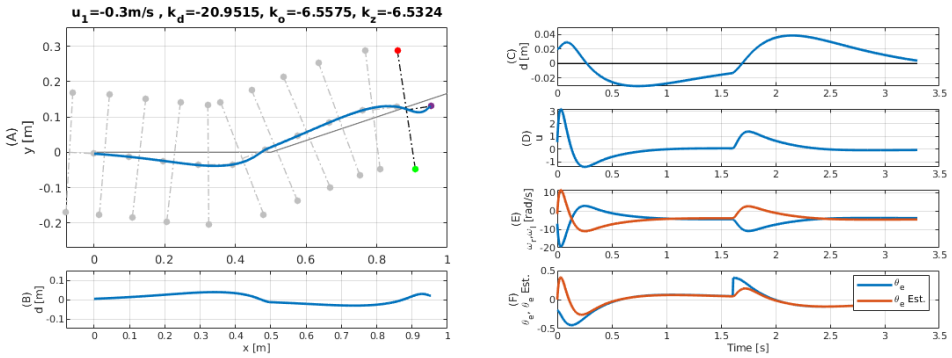

Figure 4.8: Track 2 moving backwards

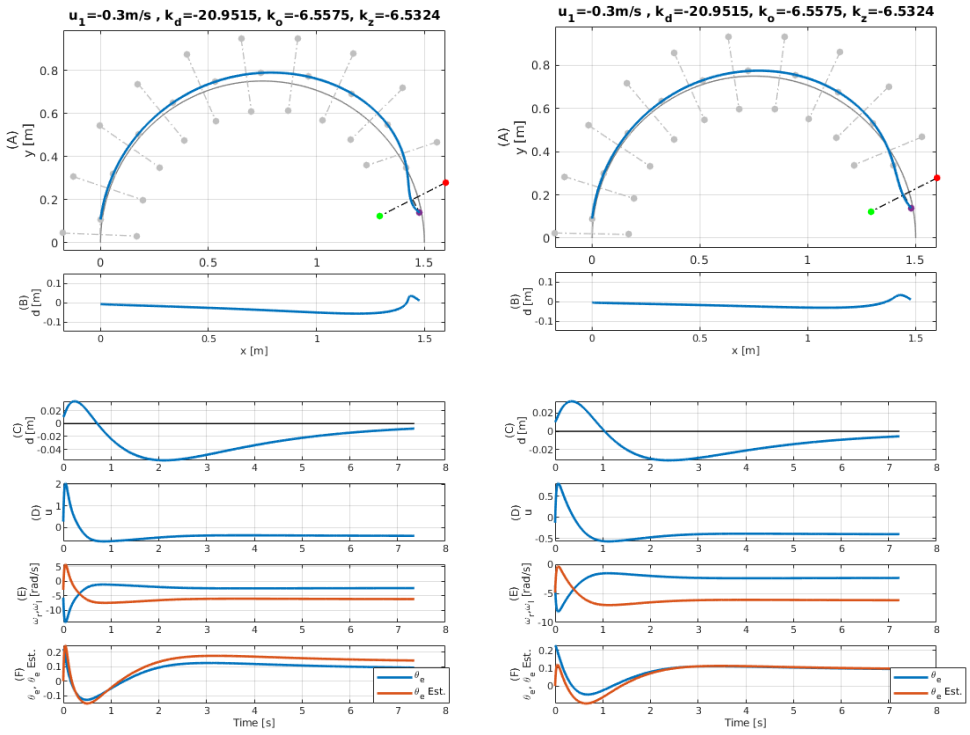

Figure 4.9: Track 3 with radius $r = 0.75m$, left unknown curvature and right $c = 1/0.75m^{-1}$

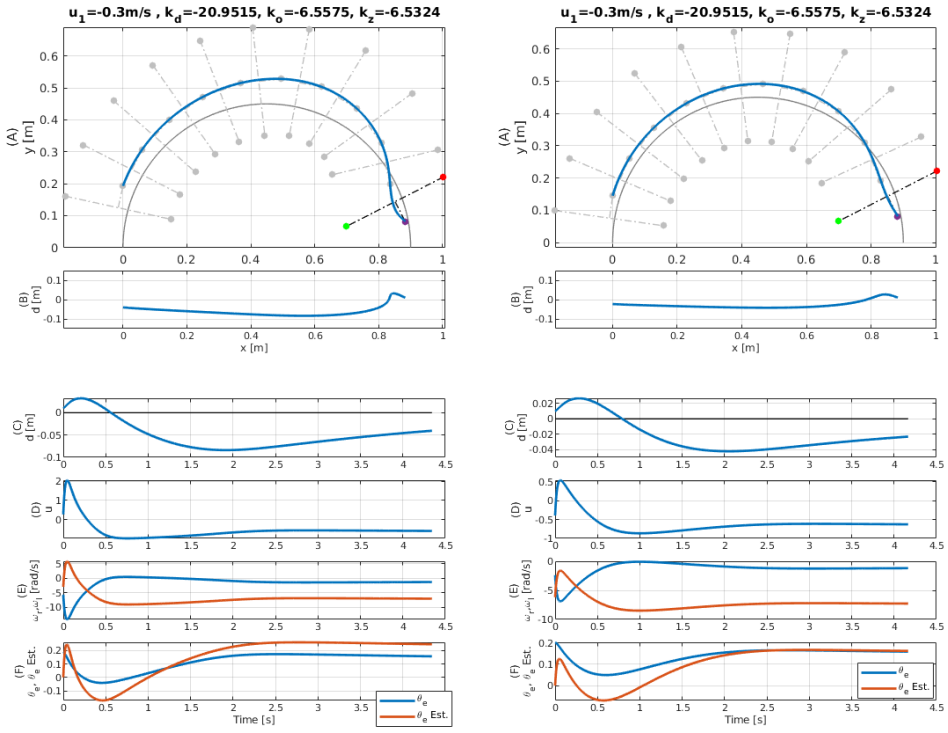

Figure 4.10: Track 3 with radius $r = 0.45m$, left unknown curvature and right $c = 1/0.45m^{-1}$

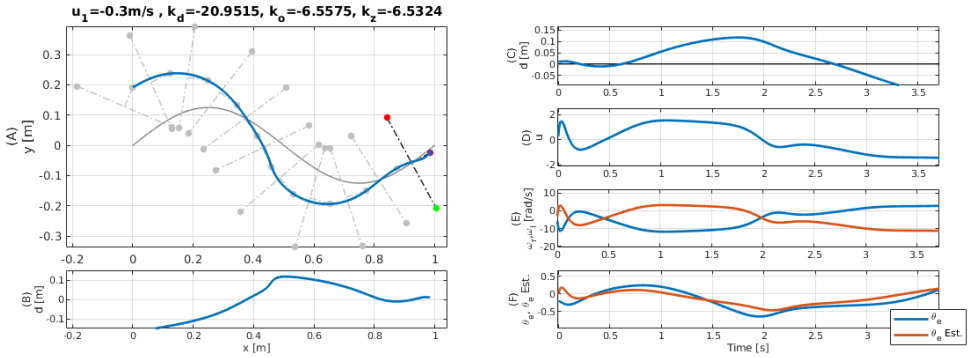

Figure 4.11: Track 4 moving backwards

in each new curvature and estimating correctly the deviation angle, see Figure 4.6.
When a negative velocity is given the deviation angle has more impact in the control algorithm, and it becomes harder to estimate. In Figures 4.9 and 4.10, is shown that when the curvature is known the estimation of the deviation angle is fast and precise, nevertheless if its unknown the estimation slowly tends to the θ_e providing a big starting error, that could produce a failure, when the control algorithm is stabilising, that error comes from the treatment of the curvature as a disturbance (3.41) and (3.40). This error turn into more detrimental if the $c \neq ct$ or the existence of a punctual disturbance, as a consequence of the continuous change of the deviation angle θ_e, Figures 4.8 and 4.11.

5 | Experimental results

Once the design was tested under simulations the implementation process started. All this process was develop in C using System Workbench for STM32, an open source software provided by Ac6 [8], based on Eclipse IDE. All the data collection shown in this section was acquired by WiFi through a python program developed simultaneously by another bachelor student [9]. The implementation was done in steps in order to notice and prevent possible errors and check safely the proposed solution.
The first step consisted on coding the state-space feedback controller without the Luenberger observer. That meant that estimated deviation angle $\widetilde{\theta}_e = 0$, in the control action suggested in (3.23):

$$u = -k_i z - k_d d \tag{5.1}$$

Being this controller the same than a proportional integral control (PI), with the difference that instead of multiplying the gain by the error of the distance d, is multiplied by its real state, state-space variables.

The state-space feedback controller works properly, however the the performance of the controller is improved when the observer is added, and all the control action could be executed. Once the implementation was achieved, the control and observer algorithms were tested in three tracks of the ones used during the simulation stage (in Track 2 was not possible to simulate the sudden change of direction, instead a 90° curve with a radius of $r = 0.15m$ was used). Additionally Track 3 is tested with the three different radius values, $r = 0.75, 0.45, 0.15m$. With this experimental tests the behaviour of the control algorithm and observer can be analysed and compared with the numerical simulations done previously. Moreover, the suggested solution is compared to the previous control algorithm, a Proportional Integral (PI) controller, for this comparison a new track (Track 5) with $c \neq ct$ is used, for this experimental work the settling time of the suggested control algorithm is set to $t_s = 1s$.

Once the controller succeed all this different real tests the vehicles velocity was changed to negative, and tested when moving backwards, knowing the constrains produced by treating the curvature as a disturbance mentioned in the previous chapter.

For all the forwards experiments, except if mentioned before, the settling time is $t_s = 3s$ and the Luenberger observer gains are $L_1 = 50$ and $L_2 = -230$.
For the backwards a settling time of $t_s = 6.5s$ and observer gains are $L_1 = 50$ and $L_2 = 1150$.

The different used tracks are shown in Figures 5.7 and 5.8.

5.1 Experimental response vs Simulations

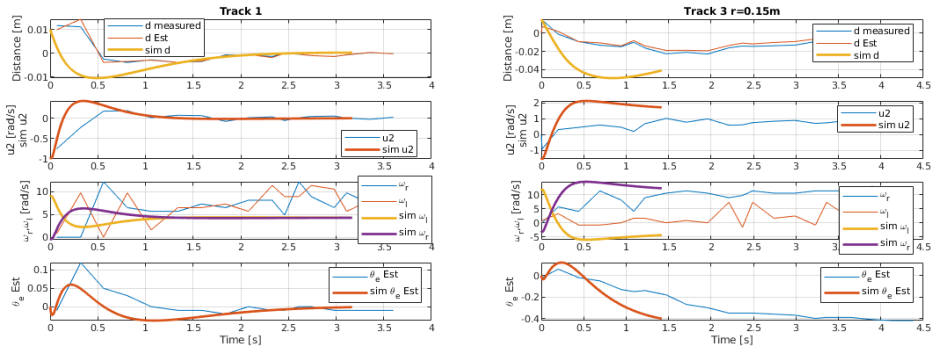

Figure 5.1: Track 1 and Track 3 radius of $0.15m$

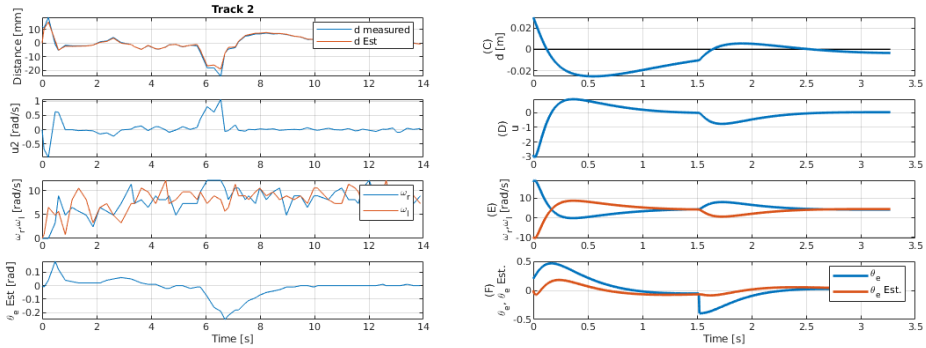

Figure 5.2: Track 2

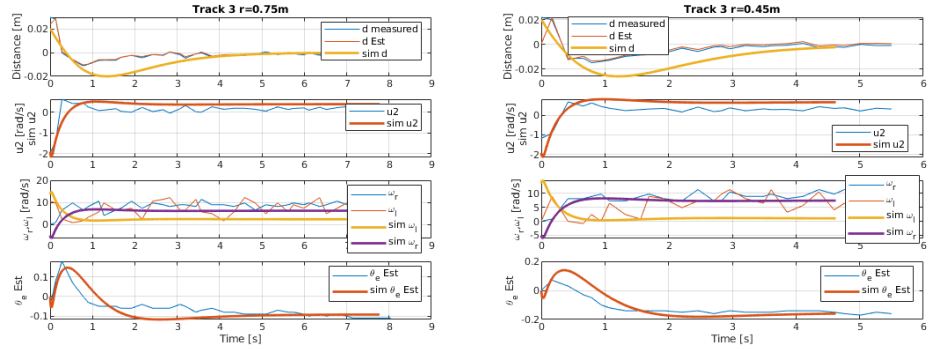

Figure 5.3: Track 3 radius of $0.75m$ and $0.45m$

Comparing the responses of all the data collected versus its simulation, the system suggested composed by a state-space feedback controller and a Luenberger observer can be definitely accepted as correct. The most important factor to check in that figures is the response of the estimated deviation angle $\widetilde{\theta}_e$, which is the differential factor added in this controller. Although, in Figure

5.1 the response when the radius $r = 0.15m$ is poorer than in the simulation, expected for the big impact produced by the curvature to the control algorithm and the existence of external noise, in all the other cases there exist a precise and fast estimation, even when it is suddenly modified Figure 5.2.

5.2 State-space feedback controller with observer vs PI

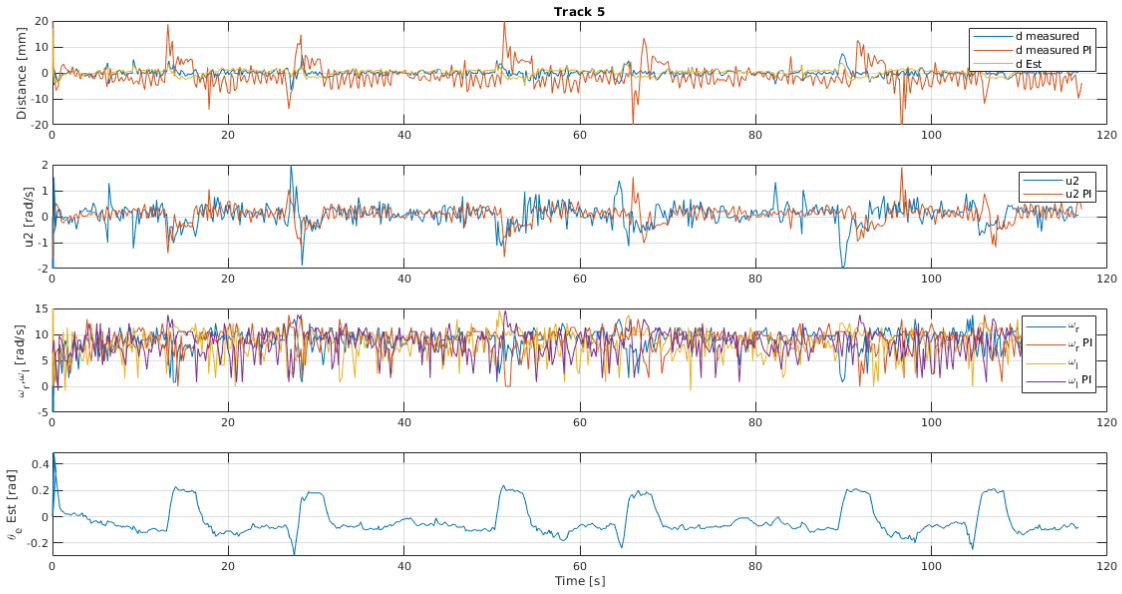

Figure 5.4: State-space control with a Luenberger observer vs PI

The comparison between the suggested space-state feedback controller with a Luenberger observer and the previous PI controller, clearly shows the response enhancement. The oscillations of error (distance from the line) during the whole path when using a PI are wider than with the suggested system. The PI suffers with high curvatures and the sudden change of them, generating a remarkable error, while the state-feedback control suffers less. This faster adjustment and decrease of error is on account of the estimated deviation angle $\widetilde{\theta}_e$, which is continuously pursuing the real angle θ_e and affecting the control action to correct its trajectory.

5.3 Backwards response

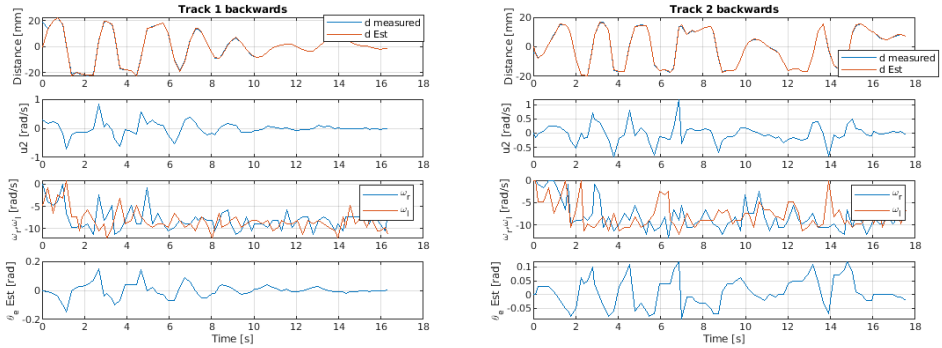

Figure 5.5: Track 1 and Track 2 backwards

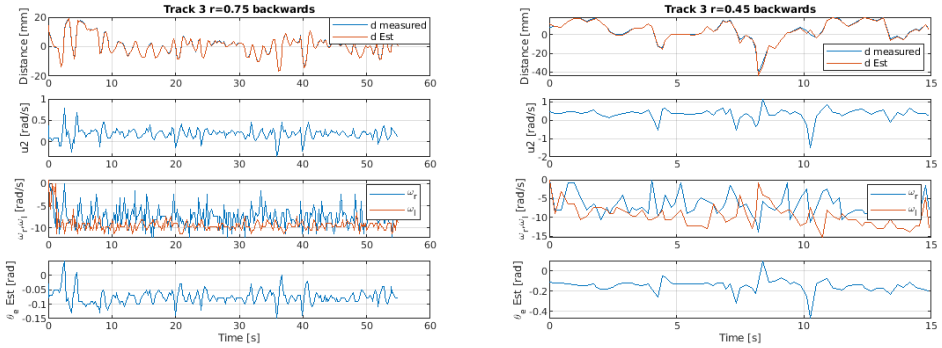

Figure 5.6: Track 3 backwards radius of $0.75m$ and radius $0.45m$

When the robot moves backwards, the performance of the respose clearly decreases. However the robot manage to follow tracks in which its curvature is constant $c = ct$. The suggested control algorithm suffers to settle into the stationary state, as consequence of the difficulty for the Luenberger observer to estimate and stabilise at the desired deviation angle θ_e^*. The issues of the control algorithm when facing a sudden change of the curvature, Figure 5.5, are consequence of slow estimation. This makes that the controller algorithm is not able to follow a not constant curvature path in which the changes of the curvature are high and fast, as the track shown in Figure 5.8 (right). This error could be reduced by treating the curvature as a state variable instead than as a disturbance.

5.4 Used tracks

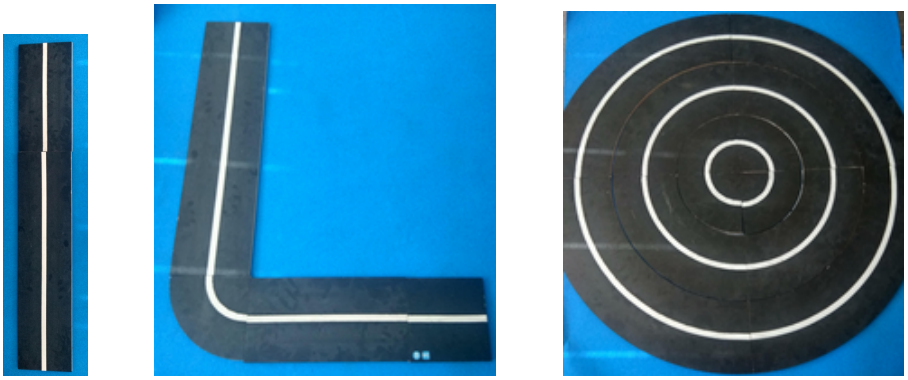

Figure 5.7: Track 1 (left), Track 2 (middle) and Track 3 (right)

Figure 5.8: Track 2 for backwards motion (left) and Track 5 (right)

6 | Environmental impact

This End of Bachelor's Project concerns in a research work that is not related with the environment. Consequently, I did not considered any direct environmental impact as consequence of the construction, installation or use of any machine tool.
The only environmental impact that could be considered during the development of this project is the electricity used by the computer and the robot and the use of paper sheets. Any measurement of the amount, or equivalent CO2 emissions does not correspond in truly assessing the environmental impact.

Conclusions

A state-feedback control algorithm (including a Luenberger observer for the deviation angle) for a differential-drive mobile robot tracking a path has been designed and experimentally tested. The obtained behaviour has been compared with a classic PI controller and two main advantages are found:

- Thanks to the observer algorithm the overall performance has been enhanced with the previous control design.
- The new control algorithm allows backwards motions when the curvatures of track are sufficiently soft.

Next and future works could be consider in better analysing how to improve the behaviour when the robot move backwards, as well as including algorithms for estimating the curvature.

Budget

In this chapter is shown the budget needed to develop this project. This costs are segmented to robot development cost and the study cost.

Robot cost

In this part is considered the cost of the different components of the robot and the assembly time, without which is not possible to develop the implementation of the control algorithm and the respective experiments.

Concept	**PVP [euros]**	**Units**	**Total Cost [euros]**
POWER			
DC/DC 12V/5V	6.30	1	6.30
DC/DC 12V/3V3	2.87	1	2.87
Driver	7.00	1	7.00
Motor	28.98	2	57.96
Battery	32.50	1	32.50
XT-60 connector	1.10	1	1.10
SIGNAL			
Distance sensor	16.47	3	49.41
Line sensor	8.46	1	8.46
Wi-Fi module	6.53	1	6.53
BOARDS			
STM32F407	21.99	1	21.99
PCB	10.83	1	10.83
MECHANICAL PARTS			
L-joint	6.30	1	6.30
Wheel	2.14	2	4.28
Wheel adaptor	5.93	2	11.86
Roller ball	6.57	1	6.57
Chassis	20.00	1	20.00
ASSEMBLY			
Laboratory technician hours	35.00	3	105.00
		TOTAL	**358.96**

Table 6.1: Robot cost. Source [10]

Study cost

The cost of the hardware and software used for the development of the paper and the engineer salary. The first table considers all the hardware and software costs, taking into account that the only non open-source software used are Matlab and Simulink. The budget considers the Academic

Use - Individual and Designated Computer annual license, for those software's.
On the other hand, the worker salary must be taken into consideration. Considering the approximate salary of a just graduated Mechanical Engineer, 10 euros per hour. And the time spent on the development of the project 600h, considering that the end of degree project is 24 ECTS and each ECTS is 25 houres.

Concept	**PVP [euros]**	**Units**	**Total Cost [euros]**
HARDWARE			
Laptop	218.99	1	218.99
SOFTWARE			
Matlab	250.00	1	250.00
Simulink	250.00	1	250.00
SALARY			
Engineer salary	10.00	600	6000.00
		TOTAL	**6718.99**

Table 6.2: Study Cost

Total cost of the project

The total cost of the research paper is of **7077.95** euros.

Bibliography

[1] Iván Prats Matinho. Control design and implementation for a line tracker vehicle. *End of Degree Project, Universitat Politècnica de Catalunya*, 2016.

[2] Antoni Riera Seguí. Disseny i implementacio d'un sistema de comunicacions wifi per a una xarxa de vehicles autonoms. *End of Degree Project, Universitat Politècnica de Catalunya*, 2016.

[3] Albert Costa Ruiz. Design of controllers and its implementation for a line tracker vehicle. *End of Degree Project, Universitat Politècnica de Catalunya*, 2017.

[4] Josep Oriol Torta Valmaña. Hardware design and implementation of the two-wheeled vehicle robots for a platooning system. *End of Degree Project, Universitat Politècnica de Catalunya*, 2017.

[5] Josep M. Olm Arnau Dorìa-Cerezo, Domingo Biel and Victor Repecho. Sliding mode control of a differential-drive mobile robot following a path. *Proc. European Control Conference*, 2019.

[6] Katsuhiko Ogata. *Modern Control Engineering*. Aeeizh, 2002.

[7] STMicroelectronics. Stm32f4 series. 2019. https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.htmloverview.

[8] Ac6. Openstm32 community: The stm32 systems resource. *Wiki*, oct 2013. http://www.openstm32.org/System%2BWorkbench%2Bfor%2BSTM32.

[9] Marc Salvadó Benasco. Optimization of the communication system and the line tracking performance for a set of robots. *End of Degree Project, Universitat Politècnica de Catalunya*, 2019.

[10] Clàudia Cabré Piqueras. Sensor ultrasònic múltiple per a conducció autònoma. *End of Degree Project, Universitat Politècnica de Catalunya*, 2019.

Annex A: Control algorithm code

The main.c code of the robot has been develop by several students in their previous project, in this project we focused in the main function, the initialisation of the state-space controller and Luenberger function and the control routine.

```
/**
  ******************************************************************************
  * File Name          : main.c
  * Description        : Main program body
  ******************************************************************************
  *
  * COPYRIGHT(c) 2016 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without
     modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include <stdio.h>
#include "stm32f4xx_hal.h"
#include "stm32f4_discovery.h"
#include <stm32f4xx_hal.h>
#include <comunicacions.h>
#include <parameters.h>
#include <math.h>

float pointer = 0;
```

```
float stage = 0;
int t1 = 0;
uint8_t icc = 0;
uint8_t iccInd = 1;
extern float desired_angle_deg;
float k_correct_angle = 2./3;
extern int desired_angle_sign;
float desired_angle_rad;
float current_angle_rad_abs = 0;
float current_angle_rad_abs_0 = 0;
float current_angle_rad_rel;
int ll = 2000;
int ticks = 0;
int ticks0 = 0;
float delta_time;
int c0 = 0; //int c1 = 0;
int c2 = 0; int c3 = 0; int c4 = 0; int c5 = 0;

/* Global variables ------------------------------------------------------- */

//COMMUNICATION
extern uint8_t paquetSortida[MIDA_MAXIMA_PAQUET];
extern BufferFIFO bufferEntradaUART;
uint8_t caracterRx;
uint8_t start = 0;


//MOTOR SPEED CONTROL
__IO float wRref=0.0;                   // Right wheel speed reference in m/s;
    wRref=2*u1-wLref
__IO float wLref=0.0;                   // Left wheel speed reference wLref=R*u2-
    u1
__IO float dutyRref;                    //Duty cycle, between 0 and 1
__IO float dutyLref;
__IO float errorR=0;                    //Speed error
__IO float errorL=0;
__IO float KpMotor=0.25;                                        //Proportional
    gain for speed control
__IO float KiMotor=0.2/30/CONTROL_FREQ;         //Integral gain, same for both
    motors
__IO float uSpeedPR=0;                  //Proportional action, right wheel
__IO float uSpeedIR=0;                  //Integral action
__IO float uSpeedFFR=0;                 //Feedforward action
__IO float uSpeedPL=0;
__IO float uSpeedIL=0;
__IO float uSpeedFFL=0;
__IO float uR=0;                                    //Voltage to be supplied to the
    motors
__IO float uL=0;
__IO float volt2duty=0;
```

```
//SPEED MEASUREMENT

__IO uint16_t encValR[2]={0,0};                                          //
    Encoder counter values
__IO uint16_t encValL[2]={0,0};
__IO uint16_t nRevR=0;
            //Number of right wheel revolutions
__IO uint16_t nRevL=0;
__IO const float speedK=SPEED_MEAS_FREQ/464.64*6.283;    //Coefficient used to get
    speed in rad/s from encoder edges count
__IO const float dacK=4095/105;
__IO float wLmeasured=0;
__IO float wRmeasured=0;

// PROXIMITY SENSOR
__IO uint32_t USValue = 0, USValue1=0, USValue2=0;
__IO float USd = 0;
            //Distance measured by the ultrasonic sensor in cm
  extern float USd_k[2];
__IO float stopDistance=7;                                               //
    Distance at which you want to stop the vehicle
__IO const float slowDistance=SLOWDISTANCE;
__IO const float slowDistanceInv=1/SLOWDISTANCE;                        //Distance
     at which you want to make the vehicle slow down
__IO float Distance_ref=30;
__IO float error_distance_ref;
__IO float Kv_dc=0.25;                                                   //
     Gain of the distance control
__IO float inv_Kv_dc=4;                                                      //
    Inverse of Kv_dc
__IO float Kp_dc=0.5;                                                    //
     Gain of the distance control

// LINE CONTROL
__IO uint16_t uhADCxConvertedValue[ADC_CHN];             //Here we store the ADC
    output
__IO float kls[4]={1,24/33.6,14.4/31.5,4.8/33.6};        //Gains used for the
    voltages coming from the line sensor
__IO float v_cruise_ref=-0.3;                                            //
    Reference cruise speed
__IO float v_cruise_max=-0.3;                                            // Maximum
     cruise speed
__IO float u1=0;
            //Cruise speed
__IO float u1_k=0;
            //Cruise speed k-1
__IO float u2=0;
            //Turning speed
```

```
__IO float d_measuredV=0.0;                                                    //
    Line Sensor Output in V
__IO float d_measured=0.0;                                                     //
    in mm
__IO float errorD=0.0;
__IO float senyPropD=0.0;                                                      //
    Proportional line control term
__IO float senyIntD=0.0;                                                       //
    Integral line control term
__IO float Kp_dist=0.25;                                                       //
    Proportional gain for line control
__IO float Ki_dist=0.0;                                                        //
    Integral gain for line control
__IO float time_ms=1.52;
__IO float filt[2];

//State-feedback controller + Observer
__IO float Kd_dist=0.0;                                                        //
    Distance gain for line control
__IO float Ko_dist=0.0;                                                        //
    Theta gain for line control
__IO float Kz_dist=0.0;                                                        //
    Integral gain state-space
__IO float theta=0.0;                                                          //
    Theta
__IO float d=0.0;
            //Distance d
__IO float z=0.0;
            //Integral action
__IO float ts = 0;
            //Established time
__IO float s1 = 0.0;                                                           //
    Real Part pole 1, 2
__IO float s3 = 0.0;                                                           //
    Real part pole 3
__IO float wd = 0.0;                                                           //
    Damped part
__IO float l = 0.07;                                                           //
    Distance from the line sensor to the wheel axes
__IO float ddhat=0;
            //Differential observed distance
__IO float dthetahat=0;                                                        //
    Differential observed deviation angle
__IO float thetaEaste=0;                                                       //
    Equilibrium deviation angle
__IO float dhat=0;
            //Observed distance
__IO int L1=0;
            //Observer gain for the distance
```

```
__IO long L2=0;
            //Observer gain for the deviation angle
__IO float dhatmm=0;                                                    //
    Observed distance in mm
__IO float c=0;
            //Curvature
__IO float d_filter[2];                                                 //
    Filter
__IO float freq=0;
            //Interruption frequency
__IO float timer=0;
            //Timer

// SLIDING MODE CONTROL

__IO float k_s=0.1;
__IO float sigma=0;
__IO float u_sliding=1,u_sliding_k=1,u_linear[2]={1,1},u_lin_sat=0;
__IO float d_measured_filter[2];
__IO float d_measured_dot=0, a=1, b=0.02, ki_linear_sl=0.2;
__IO float a_coef_aprox=0.5;

// **Other parameters** //
const int TIM1_ARR=TIM1_CK_FREQ/PWM_FREQ-1;                     //ARR value for
    timer 1, in order to have 20KHz frequency
const int TIM1_RCR=(int)(PWM_FREQ/CONTROL_FREQ)-1;              //Repetition
    counter for timer 1
const int TIM2_ARR=TIM2_CK_FREQ/(SPEED_MEAS_FREQ*SPEED_MEAS_PRES)-1;
const int TIM7_ARR=TIM2_CK_FREQ/(SPEED_MEAS_FREQ*SPEED_MEAS_PRES)-1;
const int TIM10_ARR=TIM1_CK_FREQ/(TIM10_FREQ*TIM10_PRES)-1;
const int TIM12_PSC=TIM2_CK_FREQ/TIM12_CPS-1;
const float count2us=(1000000/TIM12_CPS);        //each us corresponds to 0.02 cm
__IO float batteryV=0;                                  //Battery voltage measured
     through the voltage divider
__IO float batteryTsh=9.5;                              //Battery voltage
    threshold value, if voltage measured is lower the led turns on and driver is
    disabled


/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
DAC_HandleTypeDef hdac;
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim7;
TIM_HandleTypeDef *htim30, *htim40;
TIM_HandleTypeDef htim10;
```

```
TIM_HandleTypeDef htim11;
TIM_HandleTypeDef htim12;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM2_Init_InputCapture(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
static void MX_TIM7_Init(void);
static void MX_TIM10_Init(void);
static void MX_TIM12_Init(void);
static void MX_DAC_Init(void);

void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);

void send_channels_variables(){
        /*for(int i = 0; i < 2; i++) enviarASCII("I1sigma", 7, 15);
        for(int i = 0; i < 2; i++) enviarASCII("I2I_L", 5, 15);
        for(int i = 0; i < 2; i++) enviarASCII("I3wR_ref", 8, 15);
        for(int i = 0; i < 2; i++) enviarASCII("I4wL_ref", 8, 15);
        for(int i = 0; i < 2; i++) enviarASCII("I5wR", 4, 15);
        for(int i = 0; i < 2; i++) enviarASCII("I6wL", 4, 15);
        for(int i = 0; i < 2; i++) enviarASCII("I7dist linia", 12, 15);
        */
        uint8_t string[MIDA_PAQUET_ESTAT] = {'I'};
        char x = 'A';

        memcpy(&string[1], &x, sizeof(char));

        enviaUart(&string, MIDA_PAQUET_ESTAT);
}

int main(void)

{
        /* MCU Configuration
           ----------------------------------------------------------*/
u2=-Kz_dist*z-Kd_dist*d-Ko_dist*theta;
        /* Reset of all peripherals, Initializes the Flash interface and the
           Systick. */

        HAL_Init();

        /* Configure the system clock */
        SystemClock_Config();
```

```
/* Initialize all configured peripherals */
BSP_LED_Init(LED3);
BSP_LED_Init(LED4);
BSP_LED_Init(LED5);
BSP_LED_Init(LED6);


MX_GPIO_Init();
MX_TIM1_Init();  // PWM outputs for motor control
//MX_TIM2_Init();  // Syncrhonitzation for wheel speed measurements
MX_TIM7_Init();  // Syncrhonitzation for wheel speed measurements
MX_TIM3_Init();  // Capture timer for encoder LEFT WHEEL
MX_TIM4_Init();  // Capture timer for encoder Right WHEEL
MX_TIM10_Init(); // Generation of the trigger signal for proximity sensors

/*Uncomment this configuration for the car with single distance sensor*/

//MX_TIM12_Init(); // Capture timer for the pulse of the proximity sensors

/*Uncomment this configuration for the new car with three distance sensor
    */

MX_TIM2_Init_InputCapture(); //Capture timer for the pulses of the 3
    proximity sensors

MX_DAC_Init();
MX_DMA_Init();
MX_ADC1_Init();

HAL_TIM_PWM_Start_IT(&htim1,TIM_CHANNEL_1);
HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_2);
HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_2);
//HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_Base_Start_IT(&htim7);
HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
HAL_TIM_Encoder_Start(&htim4, TIM_CHANNEL_ALL);

HAL_TIM_PWM_Start_IT(&htim10,TIM_CHANNEL_1);

/*Uncomment this configuration for the car with single distance sensor*/
// HAL_TIM_IC_Start_IT(&htim12, TIM_CHANNEL_1);

/*Uncomment this configuration for the new car with three distance sensor
    */
 HAL_TIM_IC_Start(&htim2, TIM_CHANNEL_1);
 HAL_TIM_IC_Start(&htim2, TIM_CHANNEL_2);
 HAL_TIM_IC_Start(&htim2, TIM_CHANNEL_3);
```

```
HAL_DAC_Start(&hdac,DAC_CHANNEL_1);

TIM1->CCR1=(uint16_t) TIM1_ARR*0.5;                    //This is to reset
     the speed of the wheels
TIM1->CCR2=(uint16_t) TIM1_ARR*0.5;

HAL_Delay(1000); //Esperar que arranqui l'esp8266 o saltaran errors del
    port UART

//COMMUNICATION SOFTWARE INITIALIZATION
BSP_LED_On(LED4);
inicialitzarUart();
rebreUartIT(&caracterRx, 1);
configurarConnexioWifi();
establirConnexio();
HAL_Delay(100);
inicialitzaBufferFIFO(&bufferEntradaUART); /* Una altra vegada */
BSP_LED_Off(LED4);
start=0;
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_13,GPIO_PIN_RESET);
//send_channels_variables();

//Initialization of State-space Feedback Controller and Luenberger
    Observer
state_control_observer_init(&L1, &L2, ts, wd, c, s1, s3, &Kd_dist, &
    Ko_dist, &Kz_dist, &thetaEaste, v_cruise_ref, l);

// **INFINITE LOOP*/
while (1)
{

    COM_gestionarComunicacions();
    HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_12);

    if (batteryV<batteryTsh)
        {
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);   // If
                     the battery voltage is too low, turn the LED on
                HAL_GPIO_WritePin(GPIOD,GPIO_PIN_11,GPIO_PIN_RESET); //
                    and disable the driver

        }
        else
        {
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOD,GPIO_PIN_11,GPIO_PIN_SET);   //
                    Enable the driver
```

```
                }
                HAL_GPIO_WritePin(GPIOD,GPIO_PIN_11,(start==1));

                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
                HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1,DAC_ALIGN_12B_R,(uint16_t)
                    TIM4->CNT*5);
        }
}

/** System Clock Configuration*/
void SystemClock_Config(void)
{
        RCC_OscInitTypeDef RCC_OscInitStruct;
        RCC_ClkInitTypeDef RCC_ClkInitStruct;

        __PWR_CLK_ENABLE();

        __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

        RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
        RCC_OscInitStruct.HSEState = RCC_HSE_ON;
        RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
        RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
        RCC_OscInitStruct.PLL.PLLM = 4;
        RCC_OscInitStruct.PLL.PLLN = 168;
        RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
        RCC_OscInitStruct.PLL.PLLQ = 4;
        HAL_RCC_OscConfig(&RCC_OscInitStruct);

        RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                                        |RCC_CLOCKTYPE_PCLK1|
                                                            RCC_CLOCKTYPE_PCLK2;
        RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
        RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
        RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
        RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
        HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);

        HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

        HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

        /* SysTick_IRQn interrupt configuration */
        HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}
/* ADC1 init function */
void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;
```

```
  /**Configure the global features of the ADC (Clock, Resolution, Data Alignment
      and number of conversion)
  */
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = ENABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = ADC_CHN;
hadc1.Init.DMAContinuousRequests = ENABLE;
hadc1.Init.EOCSelection = DISABLE;
HAL_ADC_Init(&hadc1);

/* Used ADC input Channels
 * PA1 - ADC11 - Line sensor 1
 * PA2 - ADC11 - Line sensor 2
 * PA3 - ADC11 - Line sensor 3
 * PC5 - ADC11 - Line sensor 4
 * PC0 - ADC11 - Line sensor 5
 * PC4 - ADC11 - Line sensor 6
 * PB0 - ADC11 - Line sensor 7
 * PB1 - ADC11 - Line sensor 8
 *
 * PC1 - ADC11 - Battery voltage
 *
 * PC2 - ADC12 - Motor current Sense A
 * PC3 - ADC13 - Motor current Sense B
 *  */

/**Configure for the selected ADC regular channel its corresponding rank in the
    sequencer and its sample time.
 */
sConfig.Channel = ADC_CHANNEL_1; // PA1
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

/**Configure for the selected ADC regular channel its corresponding rank in the
    sequencer and its sample time.
 */
sConfig.Channel = ADC_CHANNEL_2; // PA2
sConfig.Rank = 2;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

/**Configure for the selected ADC regular channel its corresponding rank in the
    sequencer and its sample time.
```

```
    */
  sConfig.Channel = ADC_CHANNEL_3; // PA3
  sConfig.Rank = 3;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure for the selected ADC regular channel its corresponding rank in the
      sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_15; // PC5
  sConfig.Rank = 4;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure for the selected ADC regular channel its corresponding rank in the
      sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_10; // PC0
  sConfig.Rank = 5;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure for the selected ADC regular channel its corresponding rank in the
      sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_14; // PC4
  sConfig.Rank = 6;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure for the selected ADC regular channel its corresponding rank in the
      sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_8; // PB0
  sConfig.Rank = 7;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure for the selected ADC regular channel its corresponding rank in the
      sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_9; // PB1
  sConfig.Rank = 8;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure for the selected ADC regular channel its corresponding rank in the
      sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_11; // PC1
  sConfig.Rank = 9;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);


}
/* TIM1 init function */
```

```
static void MX_TIM1_Init(void)
{

        TIM_ClockConfigTypeDef sClockSourceConfig;
        TIM_MasterConfigTypeDef sMasterConfig;
        TIM_OC_InitTypeDef sConfigOC;
        TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig;

        htim1.Instance = TIM1;
        htim1.Init.Prescaler = 0;
        htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
        htim1.Init.Period = TIM1_ARR;
        htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
        htim1.Init.RepetitionCounter = TIM1_RCR;
        HAL_TIM_Base_Init(&htim1);

        sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
        HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig);
        HAL_TIM_PWM_Init(&htim1);

        sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
        sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
        HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig);

        sConfigOC.OCMode = TIM_OCMODE_PWM1;
        sConfigOC.Pulse = TIM1_ARR*0.5;
        sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
        sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
        sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
        sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
        sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
        HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1);

        sConfigOC.OCMode = TIM_OCMODE_PWM1;
        HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2);

        sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
        sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
        sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
        sBreakDeadTimeConfig.DeadTime = 0;
        sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
        sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
        sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
        HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig);

        HAL_TIM_MspPostInit(&htim1);

}
/* TIM2 init function */
static void MX_TIM2_Init(void)
```

```
{
        TIM_ClockConfigTypeDef sClockSourceConfig;
        TIM_MasterConfigTypeDef sMasterConfig;

        htim2.Instance = TIM2;
        htim2.Init.Prescaler = SPEED_MEAS_PRES-1;
        htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
        htim2.Init.Period = TIM2_ARR;
        htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
        HAL_TIM_Base_Init(&htim2);

        sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
        HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig);

        sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
        sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
        HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig);
}
/* TIM3 init function - ENCODER LEFT WHEEL*/
static void MX_TIM3_Init(void)
{
  TIM_Encoder_InitTypeDef sConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = ENCODER_ARR;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
  sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
  sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
  sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
  sConfig.IC1Filter = 0;
  sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
  sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
  sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
  sConfig.IC2Filter = 0;
  HAL_TIM_Encoder_Init(&htim3, &sConfig);

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig);
}
/* TIM4 init function - ENCODER RIGHT WHEEL*/
static void MX_TIM4_Init(void)
{

  TIM_Encoder_InitTypeDef sConfig;
  TIM_MasterConfigTypeDef sMasterConfig;
```

```
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 0;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = ENCODER_ARR;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
  sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
  sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
  sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
  sConfig.IC1Filter = 0;
  sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
  sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
  sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
  sConfig.IC2Filter = 0;
  HAL_TIM_Encoder_Init(&htim4, &sConfig);

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig);

}

/*TIM6 init function */
static void MX_TIM7_Init(void)
{
        TIM_ClockConfigTypeDef sClockSourceConfig;
        TIM_MasterConfigTypeDef sMasterConfig;

        htim7.Instance = TIM7;
        htim7.Init.Prescaler = SPEED_MEAS_PRES-1;
        htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
        htim7.Init.Period = TIM7_ARR;
        htim7.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
        HAL_TIM_Base_Init(&htim7);

        sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
        HAL_TIM_ConfigClockSource(&htim7, &sClockSourceConfig);

        sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
        sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
        HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig);
}

/* TIM10 init function */
void MX_TIM10_Init(void)
{
  TIM_OC_InitTypeDef sConfigOC;

  htim10.Instance = TIM10;
```

```
  //htim10.Init.Prescaler = TIM10_PRES-1;
  htim10.Init.Prescaler = 100-1;
  htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim10.Init.Period = TIM10_ARR;
  htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  HAL_TIM_Base_Init(&htim10);

  HAL_TIM_PWM_Init(&htim10);

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = TIM10_ARR*0.05*time_ms;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_LOW;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  HAL_TIM_PWM_ConfigChannel(&htim10, &sConfigOC, TIM_CHANNEL_1);

  HAL_TIM_MspPostInit(&htim10);
}
/* TIM12 init function */
void MX_TIM12_Init(void)
{
        TIM_ClockConfigTypeDef sClockSourceConfig;
        TIM_SlaveConfigTypeDef sSlaveConfig;
        TIM_IC_InitTypeDef sConfigIC;

        htim12.Instance = TIM12;
        htim12.Init.Prescaler = TIM12_PSC;
        htim12.Init.CounterMode = TIM_COUNTERMODE_UP;
        htim12.Init.Period = 65535;
        htim12.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
        HAL_TIM_Base_Init(&htim12);

        sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
        HAL_TIM_ConfigClockSource(&htim12, &sClockSourceConfig);

        HAL_TIM_IC_Init(&htim12);

        sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
        sSlaveConfig.InputTrigger = TIM_TS_TI1F_ED;//TIM_TS_TI1FP1;
        sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
        sSlaveConfig.TriggerFilter = 7;
        HAL_TIM_SlaveConfigSynchronization(&htim12, &sSlaveConfig);

        sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
        sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
        sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
        sConfigIC.ICFilter = 7;
        HAL_TIM_IC_ConfigChannel(&htim12, &sConfigIC, TIM_CHANNEL_1);
}

void MX_TIM2_Init_InputCapture(void)
```

```
{
        TIM_ClockConfigTypeDef sClockSourceConfig;
        TIM_SlaveConfigTypeDef sSlaveConfig;
        TIM_IC_InitTypeDef sConfigIC;
        TIM_MasterConfigTypeDef sMasterConfig;

        htim2.Instance = TIM2;
        htim2.Init.Prescaler = TIM12_PSC;
        htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
        htim2.Init.Period = 65535;
        htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
        HAL_TIM_Base_Init(&htim2);

        sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
        HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig);

        HAL_TIM_IC_Init(&htim2);

        sSlaveConfig.SlaveMode = TIM_SLAVEMODE_DISABLE;//TIM_SLAVEMODE_TRIGGER;//
            TIM_SLAVEMODE_RESET;
        sSlaveConfig.InputTrigger = TIM_TS_TI1F_ED;//TIM_TS_TI1FP1;
        sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
        sSlaveConfig.TriggerFilter = 7;
        HAL_TIM_SlaveConfigSynchronization(&htim2, &sSlaveConfig);

        sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
        sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
        HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig);

        sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
        sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
        sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
        sConfigIC.ICFilter = 7;
        HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_1);
        HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_2);
        HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_3);

}

/* DAC init function */
static void MX_DAC_Init(void)
{
        DAC_ChannelConfTypeDef sConfig;

        /**DAC Initialization */
        hdac.Instance = DAC;
        HAL_DAC_Init(&hdac);

        /**DAC channel OUT1 config */
        sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
```

```
        sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
        HAL_DAC_ConfigChannel(&hdac, &sConfig, DAC_CHANNEL_1);

}
/* Enable DMA controller clock*/
void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* Pins functionality */
  /*
   * PB12 --> Watchdog trigger
   * PD11 --> Driver enable
   *
   */

  /* GPIO Ports Clock Enable */
  __GPIOA_CLK_ENABLE();
  __GPIOB_CLK_ENABLE();
  __GPIOC_CLK_ENABLE();
  __GPIOD_CLK_ENABLE();
  __GPIOE_CLK_ENABLE();
  __GPIOH_CLK_ENABLE();

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* EXTI interrupt init*/
```

```
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

  /*Configure GPIO pin : PB11 PB12 PB13 PB15*/
        GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_15;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PD11, PD13, PD15*/
        GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_13|GPIO_PIN_15;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
        HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_11, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_RESET);

}

void change_lane(float* pointer, float* stage, int* icc, int* iccInd, int* start,
    float* u2, int desired_angle_sign, float current_angle_rad_abs, float*
    current_angle_rad_abs_0, float current_angle_rad_rel, float desired_angle_rad,
    float* d_measuredV)
{
        if (*iccInd){
                *stage = 1.;
                *start = 1;
                *u2 = 1.*desired_angle_sign;
                *current_angle_rad_abs_0 = current_angle_rad_abs;
                *iccInd = 0;
        }
        else if ((desired_angle_sign*(current_angle_rad_rel - desired_angle_rad)
            >= 0)&&(desired_angle_sign*(current_angle_rad_rel - desired_angle_rad -
             (*u2)*0.200)) < 0) {
                *stage = 2.;
                *u2 = 0;
        }
        else if (desired_angle_sign*(current_angle_rad_rel - desired_angle_rad -
            (*u2)*0.500) >= 0){
                *pointer = current_angle_rad_rel;
                *stage = 3.;
```

```
                *d_measuredV = 0;
                *current_angle_rad_abs_0 = 0;
                *iccInd = 1;
                *icc = 0;
        }
}

void update_d_measured(float* d_measuredV, int c0, int c2, int c3, int c4, int c5)
    {
        if (!((start)&&(!icc)&&(c0>ll)&&(c2+c3>2*ll)&&(c3+c4>2*ll)&&(c4+c5>2*ll)))
            {
                //*d_measuredV=-(kls[0]*2*(uhADCxConvertedValue[0]-
                    uhADCxConvertedValue[7])+kls[1]*1.5*(uhADCxConvertedValue[1]-
                    uhADCxConvertedValue[6])+kls[2]*1.25*(uhADCxConvertedValue[2]-
                    uhADCxConvertedValue[5])+kls[3]*(uhADCxConvertedValue[3]-
                    uhADCxConvertedValue[4]));
                //*d_measuredV=-(kls[0]*2*(uhADCxConvertedValue[0]-
                    uhADCxConvertedValue[7])+kls[2]*1.25*(uhADCxConvertedValue[2]-
                    uhADCxConvertedValue[5])+kls[3]*(uhADCxConvertedValue[3]-
                    uhADCxConvertedValue[4]));
                *d_measuredV=-(kls[0]*(uhADCxConvertedValue[0]-
                    uhADCxConvertedValue[7])+kls[1]*(uhADCxConvertedValue[1]-
                    uhADCxConvertedValue[6])+kls[2]*(uhADCxConvertedValue[2]-
                    uhADCxConvertedValue[5])+kls[3]*(uhADCxConvertedValue[3]-
                    uhADCxConvertedValue[4]))+kls[1]*1.5*(uhADCxConvertedValue[1]-
                    uhADCxConvertedValue[6]);
        } //+kls[1]*1.5*(uhADCxConvertedValue[1]-uhADCxConvertedValue[6])
}


void control_u2(float d_measuredV, float* u2){
        if (d_measuredV == 0) {
                *u2 = 0;
        }
}

void update_angles(float desired_angle_deg, float* desired_angle_rad, int* ticks0,
     float u2, float current_angle_rad_abs_0, float* current_angle_rad_abs, float*
    current_angle_rad_rel){
        float desired_angle_deg_corrected = k_correct_angle*desired_angle_deg;
        *desired_angle_rad = desired_angle_deg_corrected/180.*3.14159265358979;

        int ticks = HAL_GetTick();
        float delta_time = (ticks - *ticks0)/1000.;
        *ticks0 = ticks;

        *current_angle_rad_abs = *current_angle_rad_abs + delta_time*u2;
        *current_angle_rad_rel = *current_angle_rad_abs - current_angle_rad_abs_0;
}
```

```
//Initialization of State-space Feedback Controller and Luenberger Observer
void state_control_observer_init(int* L1, long* L2, float ts, float wd, float c,
    float s1, float s3, float* Kd_dist, float* Ko_dist, float* Kz_dist, float*
    thetaEaste, float v_cruise_ref, float l) {
        //For backwards move
                if (v_cruise_ref<0)
                        {
                        *L1=50;
                        *L2=1150;
                        ts=6.5;
                        wd=0.31;
                }
        //For forwards move
                else
                {
                        *L1=50;
                        *L2=-230;
                        ts=3;
                        wd=0.8031;
                }

                *thetaEaste=asin(-c*l);
                s1 = 4/ts;
                s3 = 5*s1;
        //Controller gains
                *Kz_dist=(s3*(powf(wd,2)+powf(s1,2)))/(v_cruise_ref);
                *Kd_dist=(-l*(*Kz_dist)+powf(wd,2)+powf(s1,2)+2*s1*s3)/(
                    v_cruise_ref);
                *Ko_dist=(*Kd_dist*l-2*s1-s3);

}

//CONTROL ROUTINE
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
        //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_13,GPIO_PIN_SET);

                        /* LINE AND DISTANCE CONTROL - Input: Line Sensor
                            Measurement, Proximity Sensor Measurement / Output:
                            Wheels Reference Speed */

                        ////////////////////////////////////////////////
                        //* If the line is white and background black *//
                        //d_measuredV=-(kls[0]*(uhADCxConvertedValue[0]-
                            uhADCxConvertedValue[7])+kls[1]*(uhADCxConvertedValue
                            [1]-uhADCxConvertedValue[6])+kls[2]*(
                            uhADCxConvertedValue[2]-uhADCxConvertedValue[5])+kls
                            [3]*(uhADCxConvertedValue[3]-uhADCxConvertedValue[4]));
                        d_measuredV=-(kls[0]*2*(uhADCxConvertedValue[0]-
                            uhADCxConvertedValue[7])+kls[1]*1.5*(
```

```
                uhADCxConvertedValue[1]-uhADCxConvertedValue[6])+kls
                [2]*1.25*(uhADCxConvertedValue[2]-uhADCxConvertedValue
                [5])+kls[3]*(uhADCxConvertedValue[3]-
                uhADCxConvertedValue[4]));
            //update_d_measured(&d_measuredV, uhADCxConvertedValue[0],
                 uhADCxConvertedValue[2], uhADCxConvertedValue[3],
                uhADCxConvertedValue[4], uhADCxConvertedValue[5]);

            ////////////////////////////////////////////////////
            /*uncomment if the line is black and the background is
                white */
            //d_measuredV=(kls[0]*(uhADCxConvertedValue[0]-
                uhADCxConvertedValue[7])+kls[1]*(uhADCxConvertedValue
                [1]-uhADCxConvertedValue[6])+kls[2]*(
                uhADCxConvertedValue[2]-uhADCxConvertedValue[5])+kls
                [3]*(uhADCxConvertedValue[3]-uhADCxConvertedValue[4]));

            //d_measured=DIST_V2MM*(d_measuredV+DIST_OFFSET);
                                //distance from the line in milimeters
            //d_measured=DIST_V2MM*(d_measuredV);
            d_measured=0.0064*(d_measuredV);
            batteryV=(float) (uhADCxConvertedValue[8]*BATTERY_K);   //
                battery voltage


            //Choice of the linear speed

//          if(USd<=stopDistance*2) {u1=0.0; BSP_LED_On(LED5);}
//          if(USd>(stopDistance*2)&&USd<=slowDistance) u1=
    v_cruise_ref*(USd*0.025);
//          else {u1=v_cruise_ref;
//          BSP_LED_Off(LED5);}

            u1=v_cruise_ref;

            /* error_distance_ref = Distance_ref  + (float)Kv_dc*u1 -
                USd;
            u1 = (float)(inv_Kv_dc*TS)*(-u1_k - (float)Kp_dc*
                error_distance_ref) + u1_k;

            if(u1>=v_cruise_max) u1=v_cruise_max;
            if(u1<=0) u1=0.0;
            if(USd<=stopDistance) u1=0.0;

            u1_k = u1; */

            //Choice of the angular speed

//
    ////////////////////////////////////////////////////////////////////////////
```

```
/* Sliding mode control with Hysteresis comparator*/
#if (__IMPLEMENT_OPTION == 1)

sigma = a*d_measured;

{
        if (sigma >= 5.0) {u_sliding = 1;}
                else if (-sigma >= 5.0) {u_sliding = -1;}
                else {
                                u_sliding = u_sliding_k;
                        }
        u2 = -k_s*(float)INV_SENSOR_DISTANCE*(u_sliding);
        u_sliding_k = u_sliding;
}
#endif


//
    ////////////////////////////////////////////////////////////////////////////

/* Sliding mode control with Hysteresis comparator*/
#if (__IMPLEMENT_OPTION == 2)

d_measured_filter[0] = d_measured_filter[1]*0.5 + d_measured*0.5;
d_measured_dot=(d_measured_filter[0]-d_measured_filter[1])*(float)PWM_FREQ
    ;
d_measured_filter[1]=d_measured_filter[0];

sigma = a*d_measured + b*d_measured_dot;

{
        if (sigma >= 10.0) {u_sliding = 1;}
                else if (-sigma >= 10.0) {u_sliding = -1;}
                else {
                                u_sliding = u_sliding_k;
                        }
        u2 = -k_s*(float)INV_SENSOR_DISTANCE*(u_sliding);
        u_sliding_k = u_sliding;
}
#endif

//
    ////////////////////////////////////////////////////////////////////////////

/* Boundary layer control with a integral action */
#if (__IMPLEMENT_OPTION == 3)

d_measured_filter[0] = d_measured_filter[1]*0.5 + d_measured*0.5;
```

```
d_measured_dot=(d_measured_filter[0]-d_measured_filter[1])*(float)PWM_FREQ
    ;
d_measured_filter[1]=d_measured_filter[0];

sigma = a*d_measured + b*d_measured_dot;

{
        if                      (sigma >= 30.0) {u_sliding = 1; u_lin_sat
            =0; u_linear[0]=1;}
        else if         (-sigma >= 30.0) {u_sliding = -1; u_lin_sat=0;
            u_linear[0]=-1;}
        else            {
                                u_sliding=0;
                                u_linear[1] = u_linear[0]+ sigma*
                                    ki_linear_sl;

                                if (u_linear[1]>1){u_lin_sat=1;}
                                else if (-u_linear[1]>1 ){u_lin_sat=-1;}
                                else {u_lin_sat=u_linear[1];}
                                }
                                u_linear[0]= u_lin_sat;
                        u2 = -k_s*(float)INV_SENSOR_DISTANCE*(u_sliding+
                            u_lin_sat);

}
#endif
//
    ////////////////////////////////////////////////////////////////////////////////

/* Approximation function */
#if (__IMPLEMENT_OPTION == 4)
{
        d_measured_filter[0] = d_measured_filter[1]*0.5 + d_measured*0.5;
        d_measured_dot=(d_measured_filter[0]-d_measured_filter[1])*(float)
            PWM_FREQ;
        d_measured_filter[1]=d_measured_filter[0];

        sigma = a*d_measured + b*d_measured_dot;

        u_sliding = sigma / (fabs(sigma)+ a_coef_aprox);
        u2 = -k_s*(float)INV_SENSOR_DISTANCE*(u_sliding);
}
#endif

//
    ////////////////////////////////////////////////////////////////////////////////

/* Linear control CARMINE */
#if (__IMPLEMENT_OPTION == 5)
{
```

```
                errorD=-d_measured;
                senyPropD = Kp_dist*errorD;
                            //Proportional control term for line control

                u2=u1*(senyPropD+senyIntD);
                timer+=TS_PWM;
        }
        #endif

        //
            ////////////////////////////////////////////////////////////////////////////////

        /* Controller + Observer Marc */
        #if (__IMPLEMENT_OPTION == 6)
        {

                freq=(float)PWM_FREQ;

//Filter
                d_filter[0] = d_filter[1]*0.5+ d_measured*0.5;
                d_measured_dot=(d_measured_filter[0]-d_measured_filter[1])*(float)
                    PWM_FREQ;
                d_filter[1]=d_filter[0];
                d=d_filter[0]/1000;

//Observer
                ddhat=-u1*cos(thetaEaste)*(theta)+l*u2+ L1*(d-dhat);
                dthetahat=-u2+L2*(d-dhat);
                dhat+=(ddhat/freq);
                theta+=(dthetahat/freq);

//Integral action
                z +=(d/freq);
                z = z*start;

//Control action
                u2=-Kz_dist*z-Kd_dist*d-Ko_dist*theta;

//Communication data
                dhatmm = dhat*1000;
                timer+=TS_PWM;
        }
        #endif


        //
                //      //else u1=v_cruise_ref*USd_k[1]*(float)inv_SLOWDISTANCE;
                //      if(u1>=v_cruise_max) u1=v_cruise_max;
                //      if(u1<=0) u1=0.0;
```

```
// Speed reference generation

wLref=(u1-WHEEL_DISTANCE*u2)*WHEEL_RADIUS_INV;          //
    Unycicle kinematics, wheels speed from cruise and
    turning speed
wRref=2*u1*WHEEL_RADIUS_INV-wLref;


//MOTOR SPEED CONTROL - Input: Wheels Reference Speed,
    Wheels Measured Speed / Output: PWM Duty Cycle

errorR=wRref-wRmeasured;                //Right wheel
    speed error
errorL=wLref-wLmeasured;                //Left wheel speed
     error

uSpeedPR=errorR*KpMotor;

                    //Propotional action
uSpeedFFR=wRref*KFF;

                    //Feedforward
if (uR<batteryV) uSpeedIR=(uSpeedIR+(KiMotor*errorR))*
    start;                      //Integral + Anti wind-up
uR=(uSpeedPR+uSpeedIR+uSpeedFFR)*start;
                                                    //Right
     wheel voltage

uSpeedPL=errorL*KpMotor;
uSpeedFFL=wLref*KFF;
if (uL<batteryV) uSpeedIL=(uSpeedIL+(KiMotor*errorL))*
    start;
uL=(uSpeedPL+uSpeedIL+uSpeedFFL)*start;

volt2duty=0.5/batteryV;                 //duty depends on
    measured battery voltage,
dutyRref=volt2duty*uR+0.5;
dutyLref=volt2duty*uL+0.5;
TIM1->CCR1=(uint16_t) TIM1_ARR*dutyRref;
TIM1->CCR2=(uint16_t) TIM1_ARR*dutyLref;

filt[1]= uhADCxConvertedValue[0]*0.01 + filt[0]*0.99;
filt[0]= filt[1];


HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1,DAC_ALIGN_12B_R,(
    uint16_t) (d_measuredV*0.2 + 2048));
```

```
                        //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_13,GPIO_PIN_RESET);

}
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)
{

}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  BSP_LED_Toggle(LED3);
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  //BSP_LED_Toggle(LED4);
  escriuBufferFIFO(&bufferEntradaUART, &caracterRx, 1);
  rebreUartIT(&caracterRx, 1);
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *UartHandle)
{
  BSP_LED_On(LED5);
}

/* USER CODE END 4 */

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
```

```
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```